Space-Efficient Manifest Contracts

Michael Greenberg
Princeton University
POPL 2015

(First-order) contracts

- **Specifications**
  - Written in **code**
  - Checked at **runtime**

(First-order) contracts

- **Specifications**
  - Written in **code**
  - Checked at **runtime**

$$\text{assert}(n \geq 0)$$

(First-order) contracts

- **Specifications**
  - Written in **code**
  - Checked at **runtime**

assert($n \geq 0$)

sqrt : $\{x : \text{Float} \mid x \geq 0\} \to \text{Float}$

Higher-order contracts

$(\{x{:}\mathrm{Int} \mid x \geq 0\} \to \{x{:}\mathrm{Int} \mid x \geq 0\}) \to \{y{:}\mathrm{Int} \mid y \geq 0\}$

You give a function *f* on Nats, I return a Nat

"even-odd rule"
— *Findler and Felleisen 2002*

Higher-order contracts

$$\underbrace{(\{x{:}\mathrm{Int} \mid x \geq 0\} \to \{x{:}\mathrm{Int} \mid x \geq 0\})}\to \{y{:}\mathrm{Int} \mid y \geq 0\}$$

You give a function *f* on Nats, I return a Nat

"even-odd rule"
— *Findler and Felleisen*
2002

Higher-order contracts

$(\{x{:}Int \mid x \geq 0\} \to \{x{:}Int \mid x \geq 0\}) \to \{y{:}Int \mid y \geq 0\}$

You give a function *f* on Nats, I return a Nat

If you don't get a Nat, oops—you blame me

"even-odd rule"
—*Findler and Felleisen*
2002

Higher-order contracts

$(\{x{:}\mathrm{Int} \mid x \geq 0\} \rightarrow \{x{:}\mathrm{Int} \mid x \geq 0\}) \rightarrow \{y{:}\mathrm{Int} \mid y \geq 0\}$

You give a function *f* on Nats, I return a Nat

If you don't get a Nat, oops—you blame me

If *f* is called with a negative number, oops—you blame me

"even-odd rule"
—*Findler and Felleisen*
2002

Higher-order contracts

$$\underbrace{(\{x{:}\mathrm{Int} \mid x \geq 0\} \to \{x{:}\mathrm{Int} \mid x \geq 0\})} \to \{y{:}\mathrm{Int} \mid y \geq 0\}$$

You give a function *f* on Nats, I return a Nat

If you don't get a Nat, oops—you blame me

If *f* is called with a negative number, oops—you blame me

If *f* returns a negative, oops—I blame you

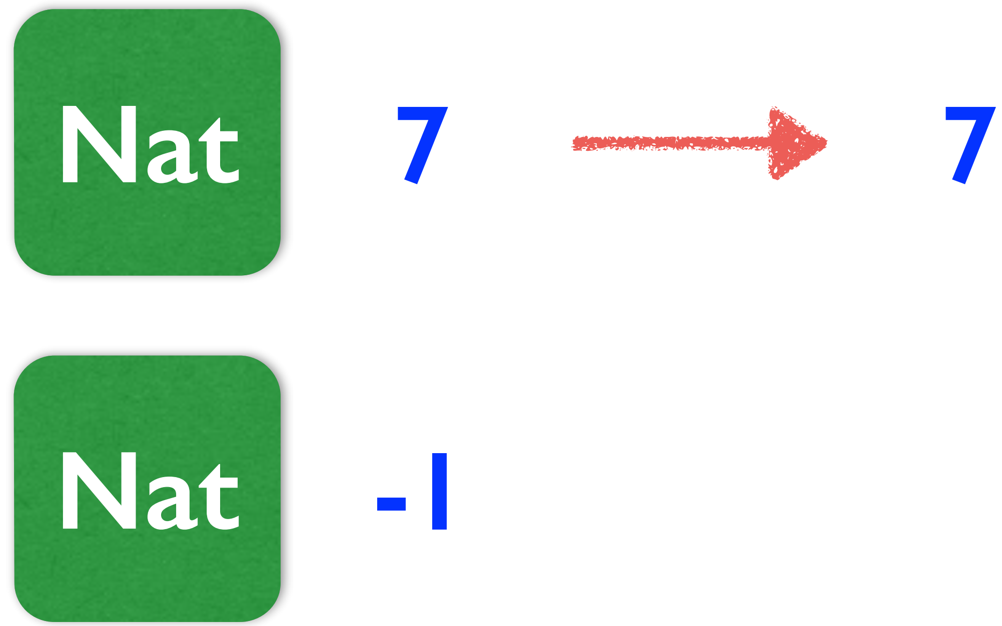
"even-odd rule"
—*Findler and Felleisen 2002*

Checking contracts at runtime

Nat 7

Checking contracts at runtime

Checking contracts at runtime

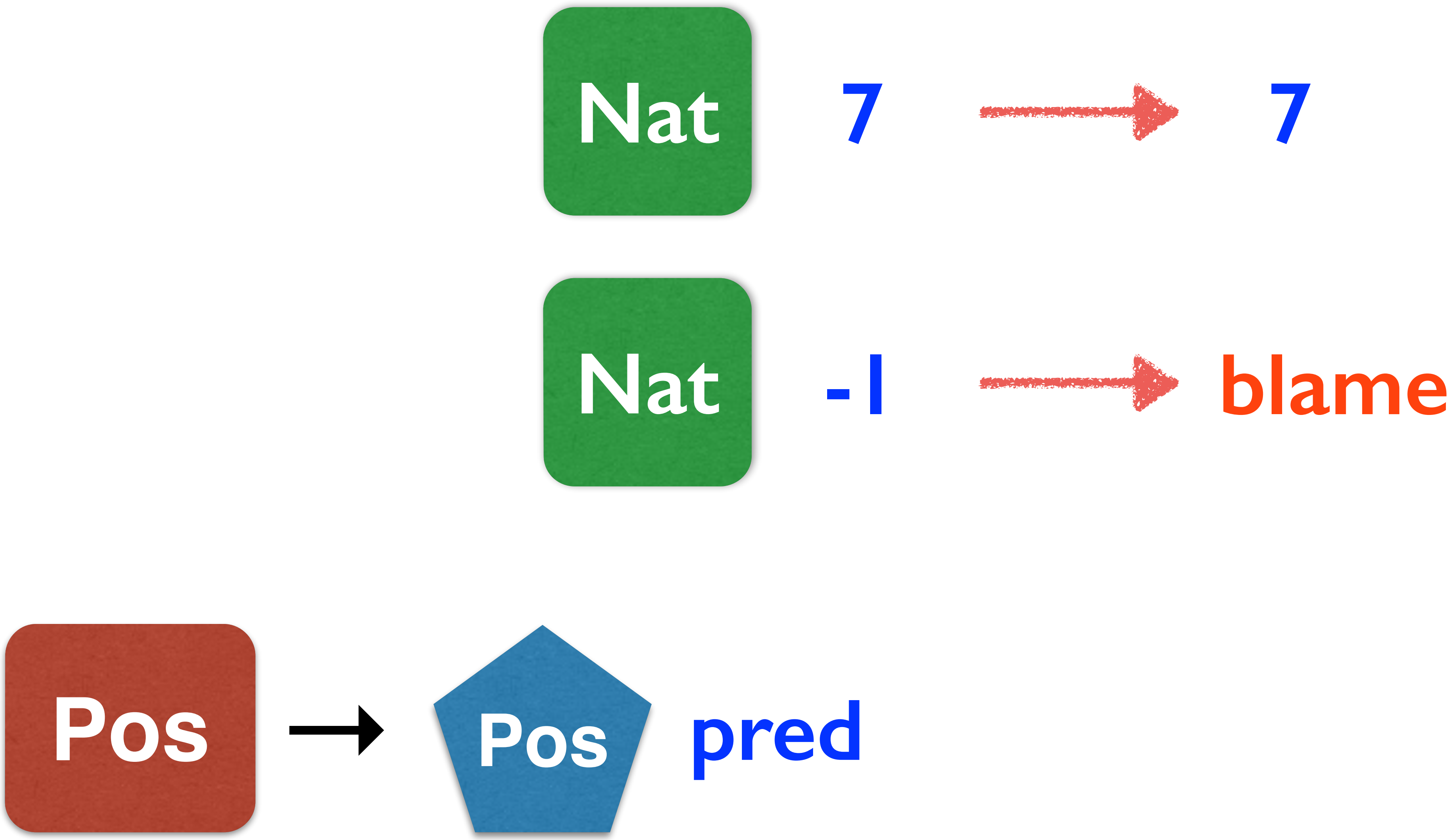


Checking contracts at runtime

Nat 7 ⟶ 7

Nat -1 ⟶ blame

Checking contracts at runtime



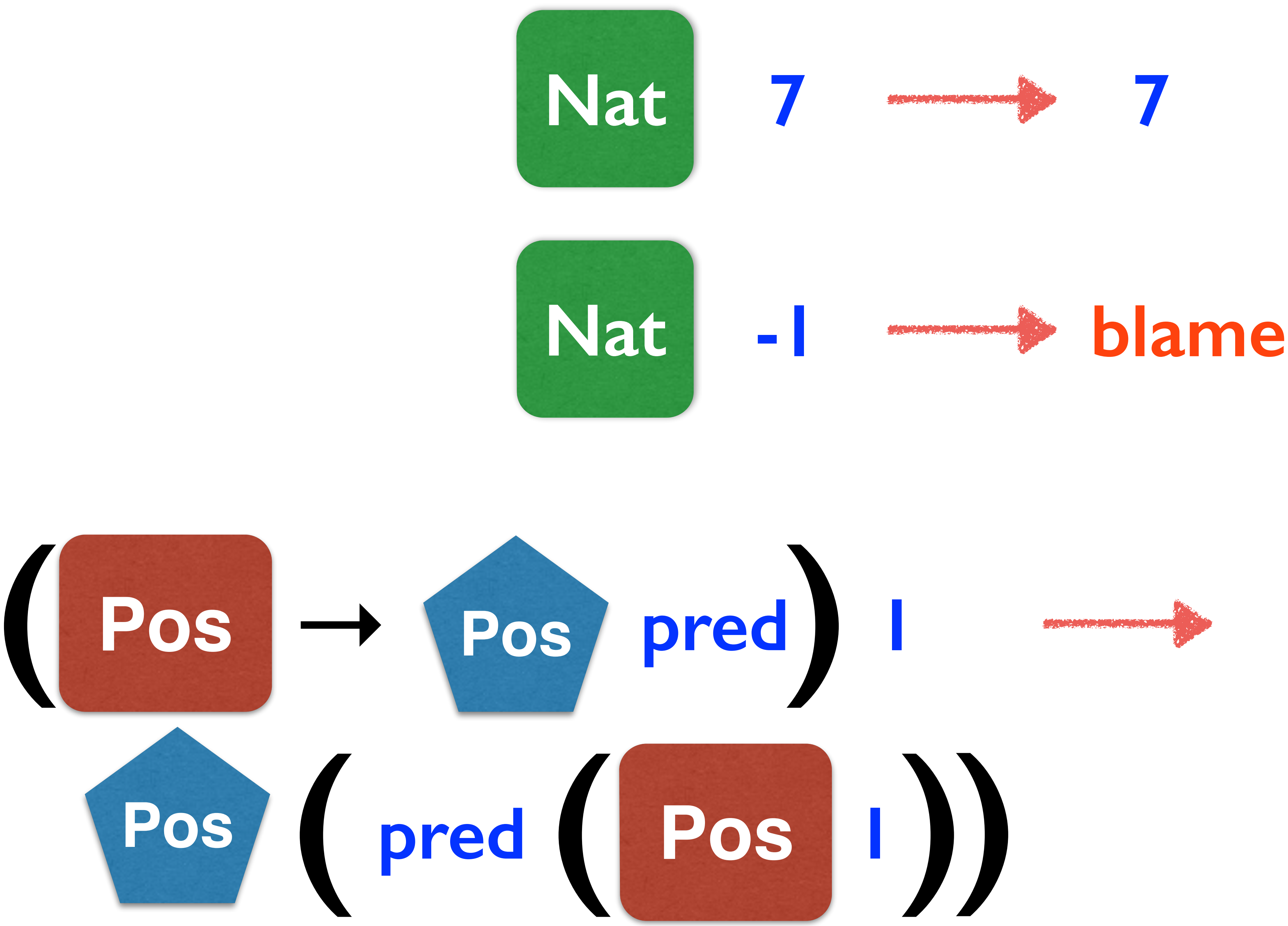
Checking contracts at runtime

Nat 7 → 7

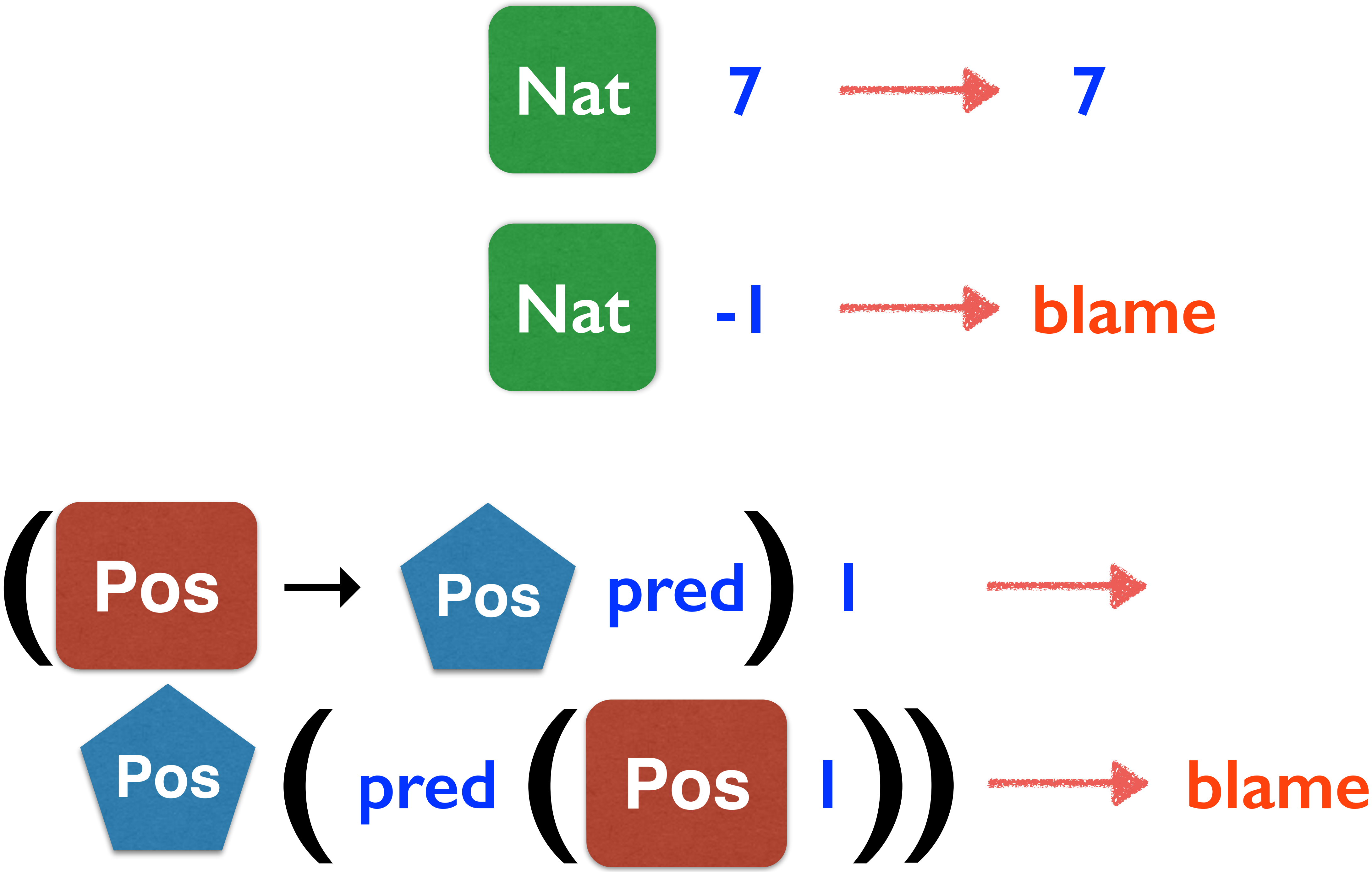
Nat -1 → blame

(Pos → Pos pred) 1

Checking contracts at runtime



Checking contracts at runtime



Bad space behavior

$$(\underline{\text{Nat} \to \text{Nat}}\ f)\ v \longrightarrow \underline{\text{Nat}}\,(f\,(\text{Nat}\ v))$$

Bad space behavior

$\underline{(\text{Nat} \rightarrow \text{Nat}\ f)}\ v$

$\rightarrow$

$\underline{\text{Nat}}\ (f\ (\text{Nat}\ v))$

My paper:
a **solution**!

Function proxies

$\boxed{\text{Set}}$

...

min : $\{l{:}\alpha\ \text{set} \mid \text{not (empty) } l\} \rightarrow \alpha$

$\boxed{\text{List}}$

...

head : $\{l{:}\alpha\ \text{list} \mid \text{not (null } l)\} \rightarrow \alpha$

Function proxies

Set

...

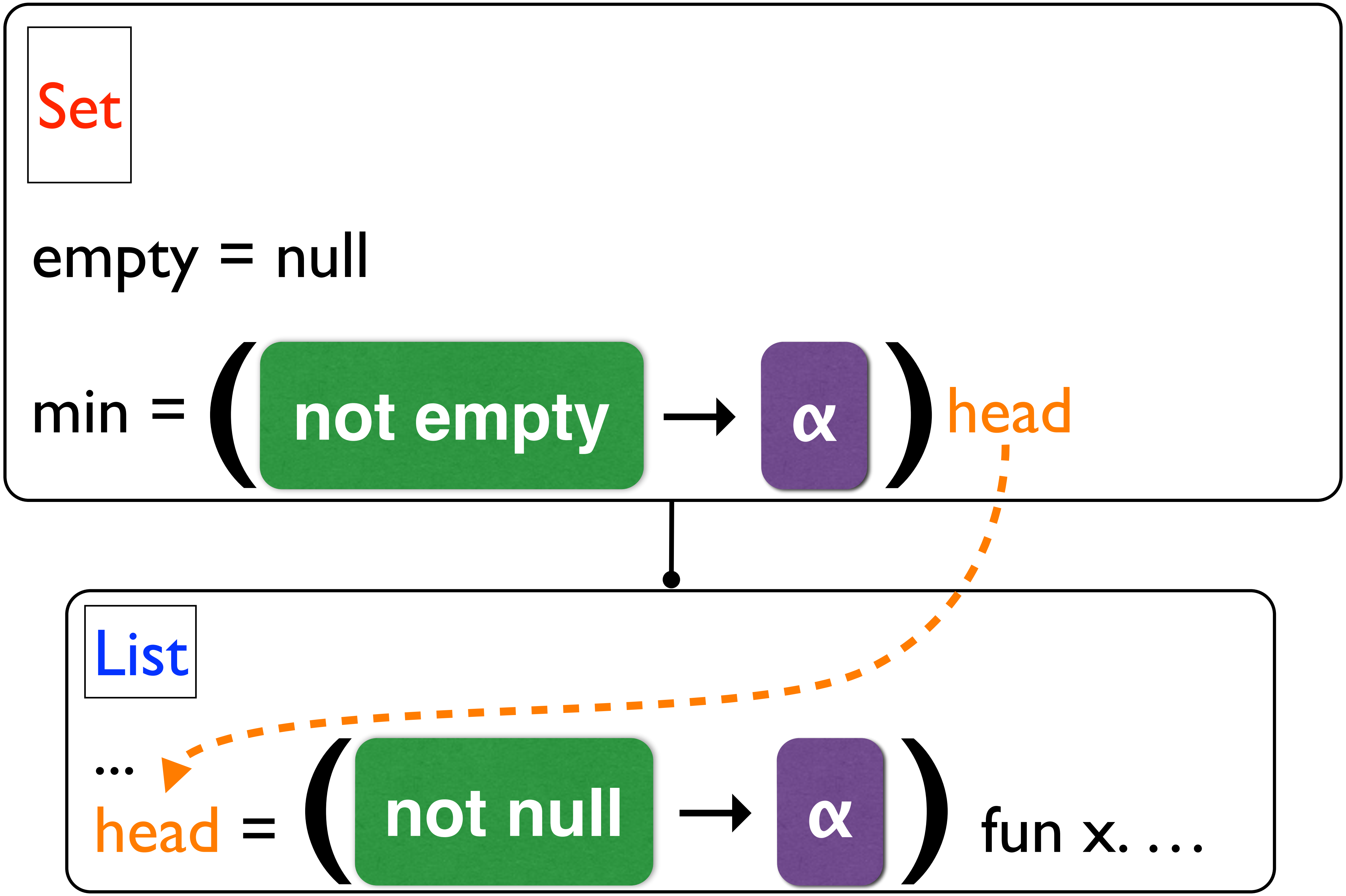
min : {l:α set| not (empty) l}→α

List

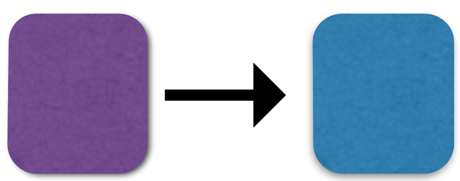
...

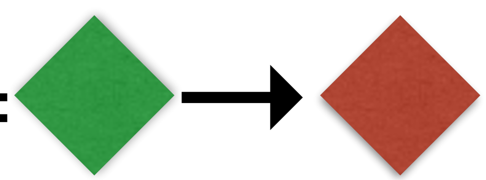
head : {l:α list| not (null l)}→α

Function proxies



Tail calls

let odd =  (λn:Int. if (n==0) then false else even (n-1))

let even =  (λn:Int. if (n==0) then true else odd (n-1))

Tail calls

let odd = → (λn:Int. if (n==0) then false else even (n-1))

let even = → (λn:Int. if (n==0) then true else odd (n-1))

Tail calls

let odd = → (λn:Int. if (n==0) then false else even (n-1))

let even = → (λn:Int. if (n==0) then true else odd (n-1))

What tail calls?

let odd = → (λn:Int. … even (n-1))

let even = → (λn:Int. … odd (n-1))

even (3)

What tail calls?

let odd = → (λn:Int. … even (n-1))

let even = → (λn:Int. … odd (n-1))

even (3)

odd (2)

What tail calls?

let odd = → (λn:Int. … even (n-1))

let even = → (λn:Int. … odd (n-1))

even (3)

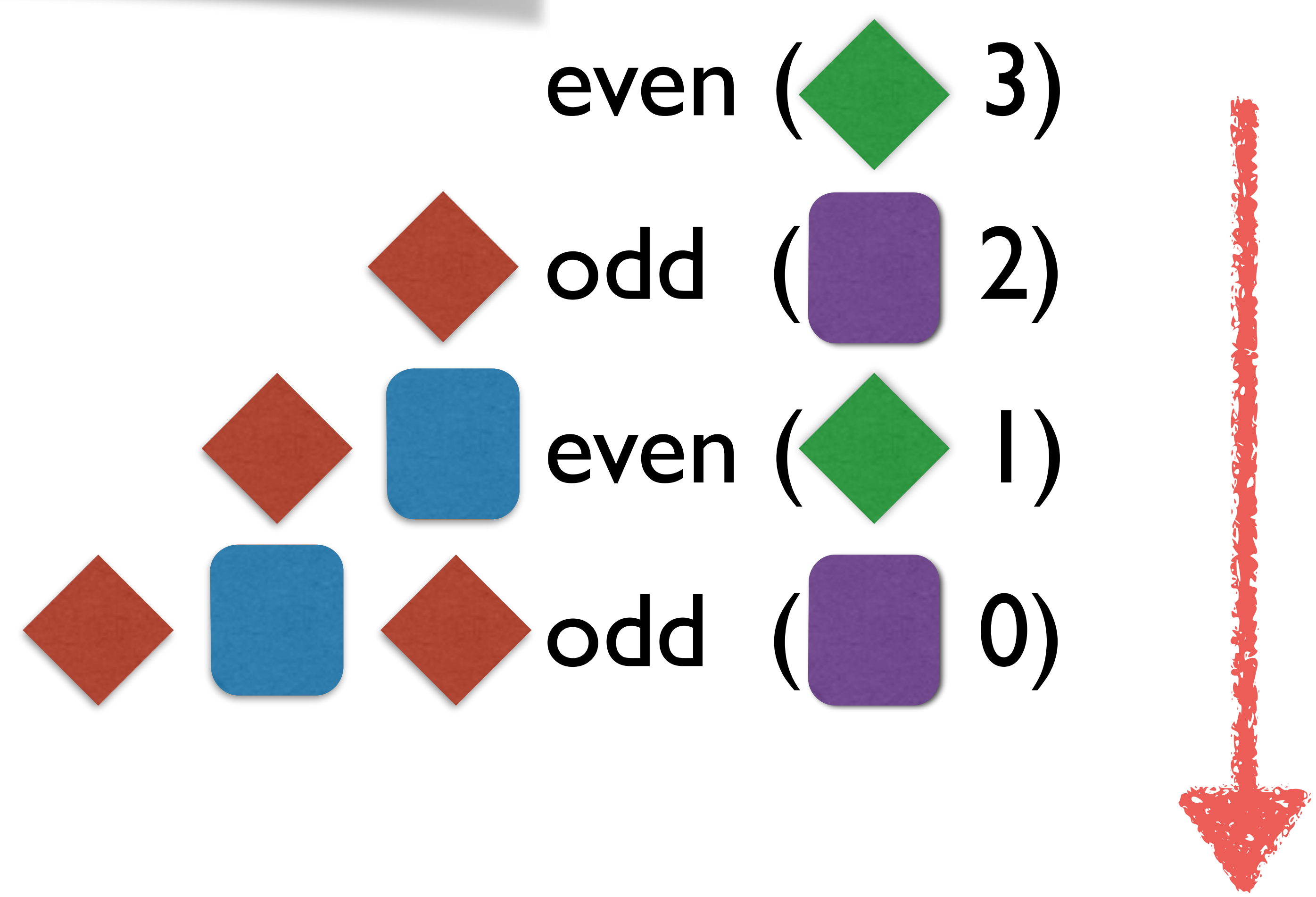
odd (2)

even (1)

What tail calls?

let odd = → (λn:Int. … even (n-1))

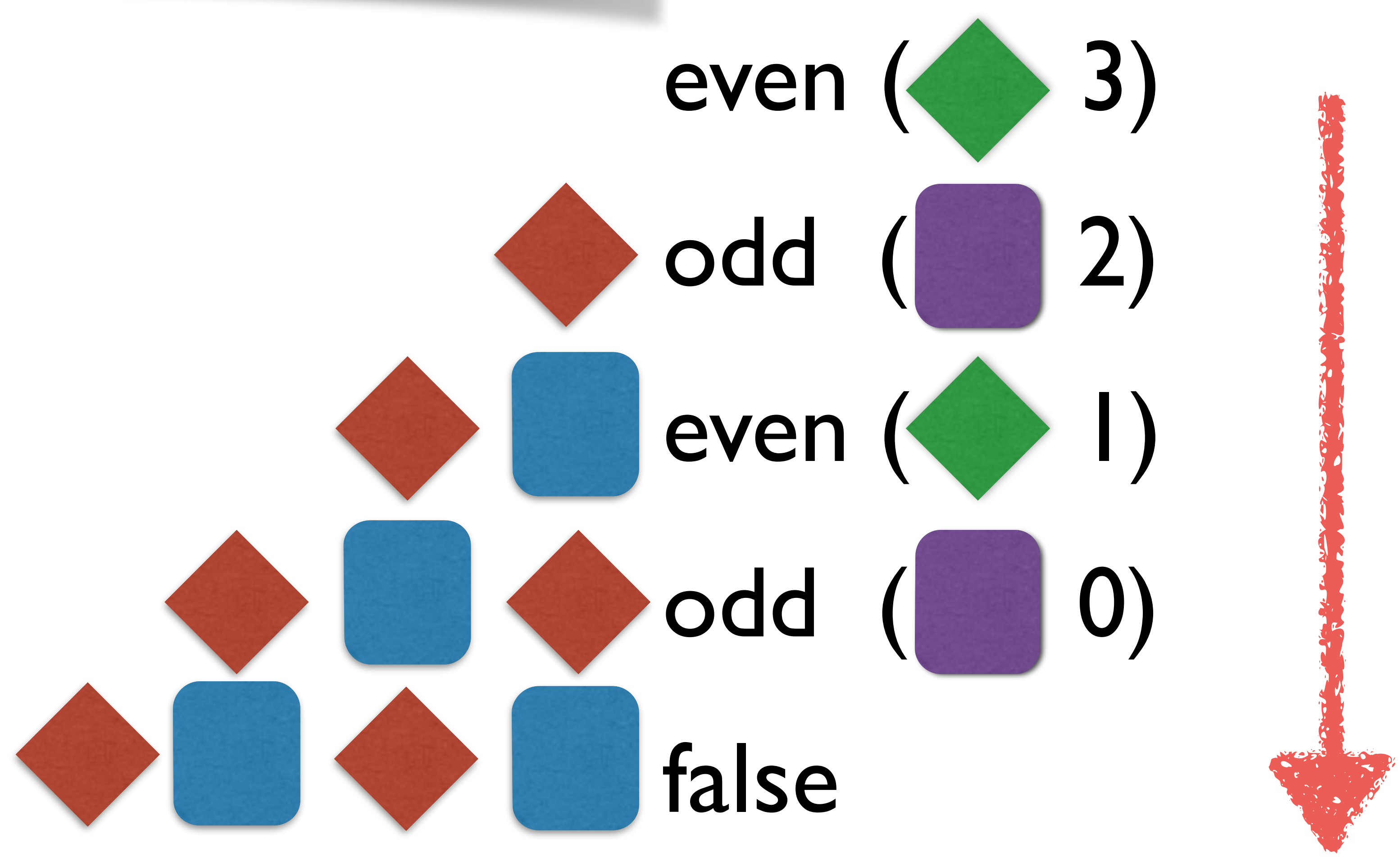
let even = → (λn:Int. … odd (n-1))



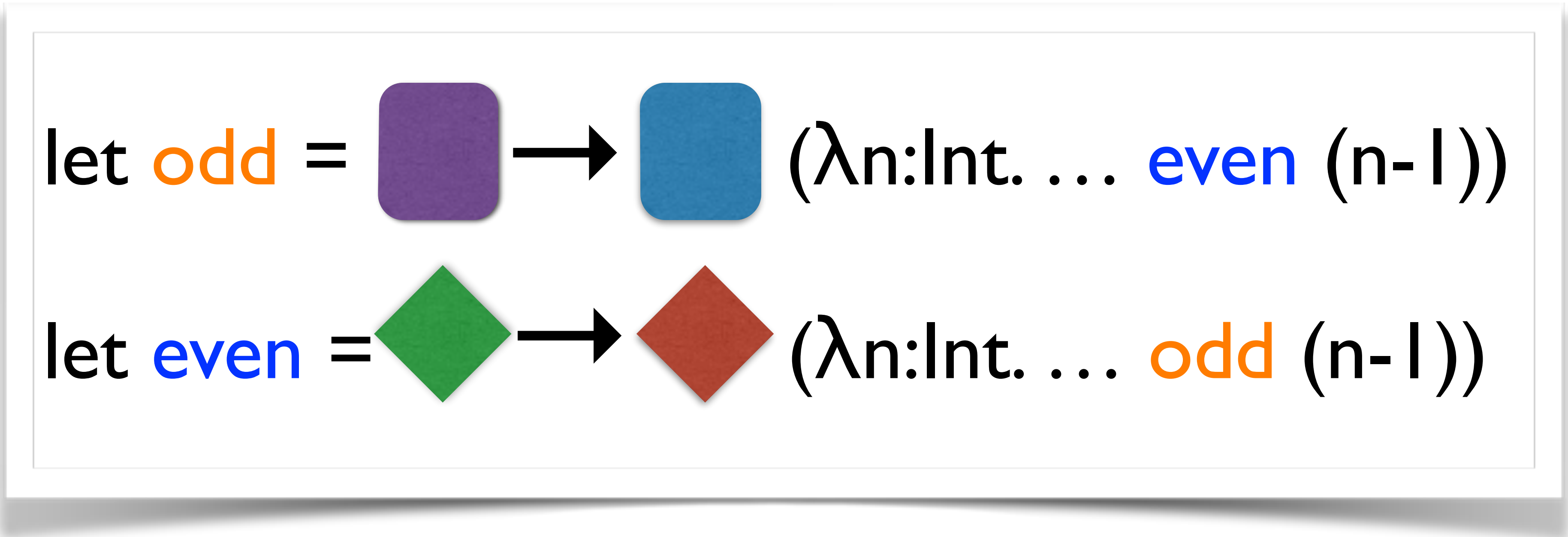
What tail calls?

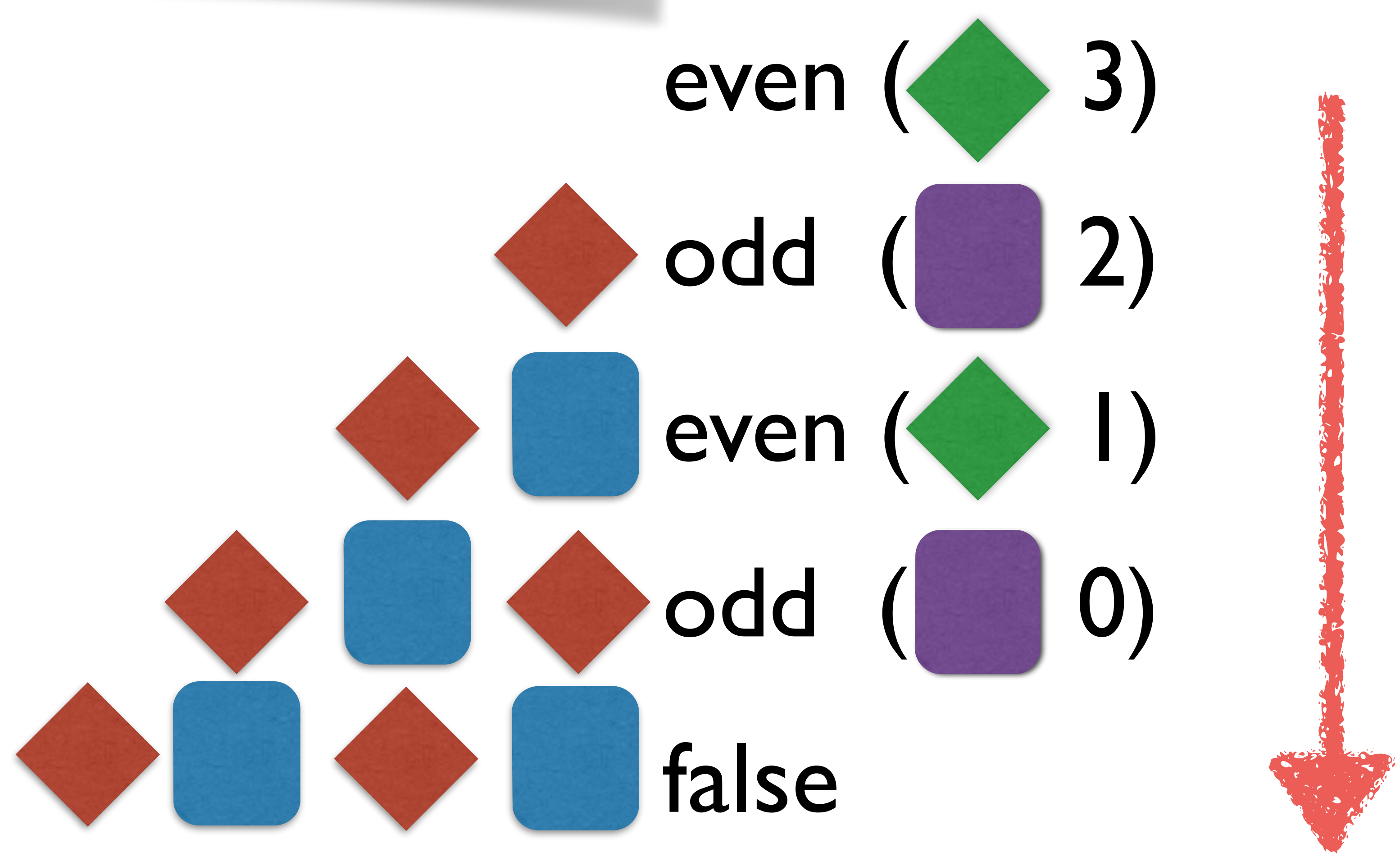
let odd = → (λn:Int. … even (n-1))

let even = → (λn:Int. … odd (n-1))



What tail calls?





Contracts break **tail calls**!

Bad space behavior

Functional Programming - Tail Calls = Bad News

- Contracts change **asymptotic space behavior**
- **Big barrier** to adoption

Space-efficient manifest contracts

a semantics for **manifest contracts**

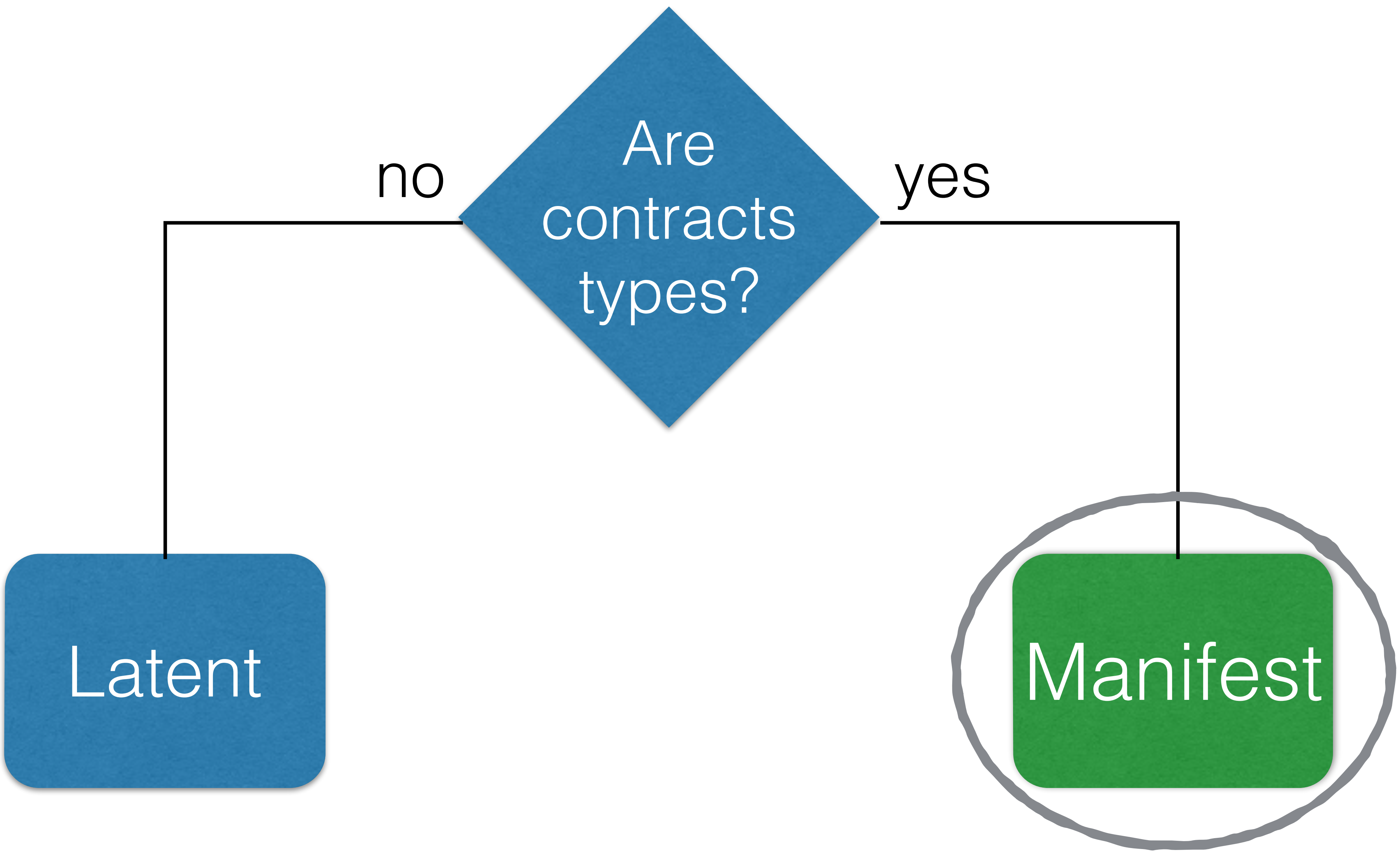
checks consume **constant** space

behave just like **classic** contracts

Westward the Course of Empire Takes Its Way
Emanuel Leutze

Contracts Made Manifest

Greenberg, Pierce, and Weirich POPL 2010



Casts

$$\langle T_1 \Rightarrow T_2 \rangle^{\ell}\ e$$

I know e has type T_1

Treat it as type T_2

If I'm wrong, blame ℓ

Casts

$$\langle T_1 \Rightarrow T_2 \rangle^{\ell}\ e$$

$$B ::= \mathrm{Bool} \mid \ldots$$

$$T ::= \{x{:}B \mid e\} \mid T_1 \to T_2$$

Casts between refinements

$$\langle\{x{:}\mathrm{Int} \mid \mathrm{true}\} \Rightarrow \{x{:}\mathrm{Int} \mid x \geq 0\}\rangle^{\ell}\ 7 \longmapsto^{*} 7$$

Casts between refinements

$\langle \{x{:}\mathrm{Int} \mid \mathrm{true}\} \Rightarrow \{x{:}\mathrm{Int} \mid x \geq 0\} \rangle^{\ell}\ 7 \longmapsto^{*} 7$

$\langle \{x{:}\mathrm{Int} \mid \mathrm{true}\} \Rightarrow \{x{:}\mathrm{Int} \mid x \geq 0\} \rangle^{\ell}\ {-1} \longmapsto^{*} \mathrm{blame}\ \ell$

Casts between functions

$\langle T_1 \to T_2 \Rightarrow U_1 \to U_2 \rangle^{\ell}\ f$

...is a **value** a/k/a **function proxy**.

Casts between functions

$$(\langle T_1 \to T_2 \Rightarrow U_1 \to U_2 \rangle^{\ell}\ f)\ v \longmapsto$$

$$\langle T_2 \Rightarrow U_2 \rangle^{\ell}\ (f\ (\langle U_1 \Rightarrow T_1 \rangle^{\ell}\ v))$$

$$(\langle \{x{:}\mathrm{Int} \mid \mathrm{true}\} \to \{x{:}\mathrm{Int} \mid \mathrm{true}\} \Rightarrow \{x{:}\mathrm{Int} \mid x \geq 0\} \to \{x{:}\mathrm{Int} \mid x \geq 0\} \rangle^{\ell}\ \lambda x{:}\{x{:}\mathrm{Int} \mid \mathrm{true}\}.\ x-1)\ 0 \longmapsto$$

$$(\langle \{x{:}\mathrm{Int}\,|\,\mathrm{true}\} \to \{x{:}\mathrm{Int}\,|\,\mathrm{true}\} \Rightarrow \{x{:}\mathrm{Int}\,|\,x \geq 0\} \to \{x{:}\mathrm{Int}\,|\,x \geq 0\}\rangle^{\ell}\ \lambda x{:}\{x{:}\mathrm{Int}\,|\,\mathrm{true}\}.\ x-1)\ 0 \longmapsto$$

$$\langle \{x{:}\mathrm{Int}\,|\,\mathrm{true}\} \Rightarrow \{x{:}\mathrm{Int}\,|\,x \geq 0\}\rangle^{\ell}\ (\lambda x{:}\{x{:}\mathrm{Int}\,|\,\mathrm{true}\}.\ x-1\ (\langle \{x{:}\mathrm{Int}\,|\,x \geq 0\} \Rightarrow \{x{:}\mathrm{Int}\,|\,\mathrm{true}\}\rangle^{\ell}\ 0)) \longmapsto^{*}$$

$$(\langle\{x{:}\mathrm{Int}|\mathrm{true}\}\to\{x{:}\mathrm{Int}|\mathrm{true}\}\Rightarrow$$

$$\{x{:}\mathrm{Int}|x\geq 0\}\to\{x{:}\mathrm{Int}|x\geq 0\}\rangle^{\ell}\ \lambda x{:}\{x{:}\mathrm{Int}|\mathrm{true}\}.\ x-1)\ 0 \longmapsto$$

$$\langle\{x{:}\mathrm{Int}|\mathrm{true}\}\Rightarrow\{x{:}\mathrm{Int}|x\geq 0\}\rangle^{\ell}$$

$$(\lambda x{:}\{x{:}\mathrm{Int}|\mathrm{true}\}.\ x-1\ (\langle\{x{:}\mathrm{Int}|x\geq 0\}\Rightarrow\{x{:}\mathrm{Int}|\mathrm{true}\}\rangle^{\ell}\ 0)) \longmapsto^{*}$$

$$\langle\{x{:}\mathrm{Int}|\mathrm{true}\}\Rightarrow\{x{:}\mathrm{Int}|x\geq 0\}\rangle^{\ell}\ (\lambda x{:}\{x{:}\mathrm{Int}|\mathrm{true}\}.\ x-1\ 0) \longmapsto^{*}$$

$$(\langle\{x{:}\mathrm{Int}|\mathrm{true}\}\rightarrow\{x{:}\mathrm{Int}|\mathrm{true}\}\Rightarrow$$

$$\{x{:}\mathrm{Int}|x\geq 0\}\rightarrow\{x{:}\mathrm{Int}|x\geq 0\}\rangle^{\ell}\ \lambda x{:}\{x{:}\mathrm{Int}|\mathrm{true}\}.\ x-1)\ 0 \longmapsto$$

$$\langle\{x{:}\mathrm{Int}|\mathrm{true}\}\Rightarrow\{x{:}\mathrm{Int}|x\geq 0\}\rangle^{\ell}$$

$$(\lambda x{:}\{x{:}\mathrm{Int}|\mathrm{true}\}.\ x-1\ (\langle\{x{:}\mathrm{Int}|x\geq 0\}\Rightarrow\{x{:}\mathrm{Int}|\mathrm{true}\}\rangle^{\ell}\ 0)) \longmapsto^{*}$$

$$\langle\{x{:}\mathrm{Int}|\mathrm{true}\}\Rightarrow\{x{:}\mathrm{Int}|x\geq 0\}\rangle^{\ell}\ (\lambda x{:}\{x{:}\mathrm{Int}|\mathrm{true}\}.\ x-1\ 0) \longmapsto^{*}$$

$$\langle\{x{:}\mathrm{Int}|\mathrm{true}\}\Rightarrow\{x{:}\mathrm{Int}|x\geq 0\}\rangle^{\ell}\ {-1} \longmapsto^{*} \mathrm{blame}\ \ell$$

$(\langle\{x{:}\mathrm{Int}|\mathrm{true}\}\rightarrow\{x{:}\mathrm{Int}|\mathrm{true}\}\Rightarrow$

$\{x{:}\mathrm{Int}|x\geq 0\}\rightarrow\{x{:}\mathrm{Int}|x\geq 0\}\rangle^{\ell}\ \lambda x{:}\{x{:}\mathrm{Int}|\mathrm{true}\}.\ x-1)\ 0 \longmapsto$

$\langle\{x{:}\mathrm{Int}|\mathrm{true}\}\Rightarrow\{x{:}\mathrm{Int}|x\geq 0\}\rangle^{\ell}$

$(\lambda x{:}\{x{:}\mathrm{Int}|\mathrm{true}\}.\ x-1\ (\langle\{x{:}\mathrm{Int}|x\geq 0\}\Rightarrow\{x{:}\mathrm{Int}|\mathrm{true}\}\rangle^{\ell}\ 0)) \longmapsto^{*}$

$\langle\{x{:}\mathrm{Int}|\mathrm{true}\}\Rightarrow\{x{:}\mathrm{Int}|x\geq 0\}\rangle^{\ell}\ (\lambda x{:}\{x{:}\mathrm{Int}|\mathrm{true}\}.\ x-1\ 0) \longmapsto^{*}$

$\langle\{x{:}\mathrm{Int}|\mathrm{true}\}\Rightarrow\{x{:}\mathrm{Int}|x\geq 0\}\rangle^{\ell}\ -1 \longmapsto^{*} \mathrm{blame}\ \ell$

Pop quiz

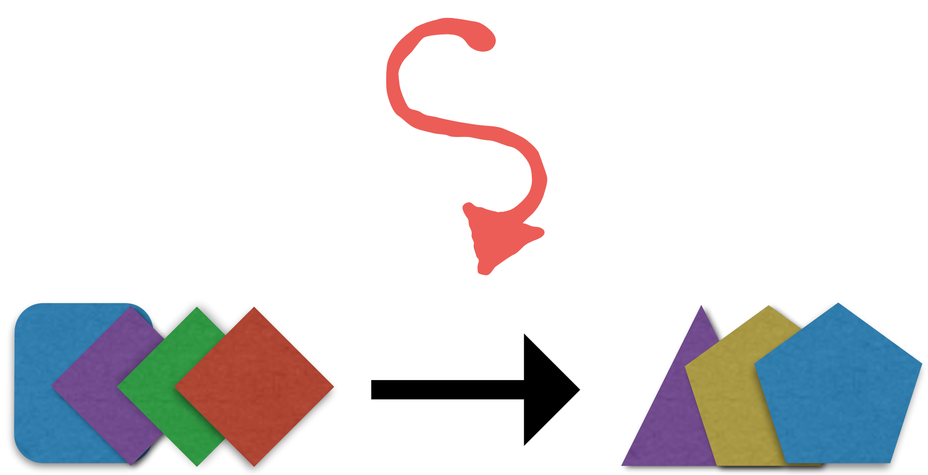
When we execute

$$<(\mathrm{Nat} \to \mathrm{Nat}) \to \mathrm{Nat} \Rightarrow (\mathrm{Pos} \to \mathrm{Pos}) \to \mathrm{Pos}>^{\ell}$$

will we check **Nat** or **Pos**
in the domain's domain?

Insight #1: use coercions

$$\langle T_1 \Rightarrow T_2 \rangle^{\ell}$$



Coercions between predicates

$\langle\{x{:}\mathrm{Int} \mid \mathrm{true}\} \Rightarrow \{x{:}\mathrm{Int} \mid x \geq 0\}\rangle^{\ell}\ 7 \longmapsto^{*} 7$

$\langle\{x{:}\mathrm{Int} \mid \mathrm{true}\} \Rightarrow \{x{:}\mathrm{Int} \mid x \geq 0\}\rangle^{\ell}\ {-1} \longmapsto^{*} \mathrm{blame}\ \ell$

Coercions between predicates

$$\langle\{x{:}\mathrm{Int} \mid \mathrm{true}\} \Rightarrow \{x{:}\mathrm{Int} \mid x \geq 0\}\rangle^{\ell}\ 7 \longmapsto^{*} 7$$

$$\langle\{x{:}\mathrm{Int} \mid \mathrm{true}\} \Rightarrow \{x{:}\mathrm{Int} \mid x \geq 0\}\rangle^{\ell}\ {-1} \longmapsto^{*} \mathrm{blame}\ \ell$$

Totally ignored!

Coercions between predicates

$\langle\{x:\mathrm{Int} \mid \mathrm{true}\} \Rightarrow \{x:\mathrm{Int} \mid x \geq 0\}\rangle^{\ell}\ 7 \longmapsto^{*} 7$

$\langle\{x:\mathrm{Int} \mid \mathrm{true}\} \Rightarrow \{x:\mathrm{Int} \mid x \geq 0\}\rangle^{\ell}\ {-1} \longmapsto^{*} \mathrm{blame}\ \ell$

Nat

Coercions between functions

$$(<\{x{:}\mathrm{Int}\mid\mathrm{true}\}\rightarrow\{x{:}\mathrm{Int}\mid\mathrm{true}\}\Rightarrow\{x{:}\mathrm{Int}\mid x\geq 0\}\rightarrow\{x{:}\mathrm{Int}\mid x\geq 0\}>^{\ell}\ f)\ v$$

$$\longmapsto$$

$$<\{x{:}\mathrm{Int}\mid\mathrm{true}\}\Rightarrow\{x{:}\mathrm{Int}\mid x\geq 0\}>^{\ell}\ (f\ (<\{x{:}\mathrm{Int}\mid x\geq 0\}\Rightarrow\{x{:}\mathrm{Int}\mid\mathrm{true}\}>^{\ell}\ v))$$

Coercions between functions

$$(<\{x{:}\mathrm{Int} \mid \mathrm{true}\} \to \{x{:}\mathrm{Int} \mid \mathrm{true}\} \Rightarrow \{x{:}\mathrm{Int} \mid x \geq 0\} \to \{x{:}\mathrm{Int} \mid x \geq 0\}>^{\ell}\ f)\ v$$

$$\longmapsto$$

$$<\{x{:}\mathrm{Int} \mid \mathrm{true}\} \Rightarrow \{x{:}\mathrm{Int} \mid x \geq 0\}>^{\ell}\ (f\ (<\{x{:}\mathrm{Int} \mid x \geq 0\} \Rightarrow \{x{:}\mathrm{Int} \mid \mathrm{true}\}>^{\ell}\ v))$$

Nat

Coercions between functions

$$(\langle \{x{:}\mathrm{Int} \mid \mathrm{true}\} \to \{x{:}\mathrm{Int} \mid \mathrm{true}\} \Rightarrow \{x{:}\mathrm{Int} \mid x \geq 0\} \to \{x{:}\mathrm{Int} \mid x \geq 0\}\rangle^{\ell}\ f)\ v$$

$$\longmapsto$$

$$\langle \{x{:}\mathrm{Int} \mid \mathrm{true}\} \Rightarrow \{x{:}\mathrm{Int} \mid x \geq 0\}\rangle^{\ell}\ (f\ (\langle \{x{:}\mathrm{Int} \mid x \geq 0\} \Rightarrow \{x{:}\mathrm{Int} \mid \mathrm{true}\}\rangle^{\ell}\ v))$$

Nat

Int

Coercions between functions

$$(\langle \{x{:}Int \mid true\} \to \{x{:}Int \mid true\} \Rightarrow \{x{:}Int \mid x \geq 0\} \to \{x{:}Int \mid x \geq 0\} \rangle^{\ell}\ f)\ v$$

$$\longmapsto$$

$$\langle \{x{:}Int \mid true\} \Rightarrow \{x{:}Int \mid x \geq 0\} \rangle^{\ell}\ (f\ (\langle \{x{:}Int \mid x \geq 0\} \Rightarrow \{x{:}Int \mid true\} \rangle^{\ell}\ v))$$

Int → Nat

Coercions between functions

$$\langle T_1 \to T_2 \Rightarrow U_1 \to U_2 \rangle^{\ell}$$

$$\langle U_1 \Rightarrow T_1 \rangle^{\ell} \qquad \langle T_2 \Rightarrow U_2 \rangle^{\ell}$$

$$\_\_\_\_ \quad \to \quad \_\_\_\_$$

Coercions between functions

$$\langle T_1 \rightarrow T_2 \Rightarrow U_1 \rightarrow U_2 \rangle^{\ell}$$

$$\langle U_1 \Rightarrow T_1 \rangle^{\ell} \qquad \langle T_2 \Rightarrow U_2 \rangle^{\ell}$$

$$\_\_\_\_ \rightarrow \_\_\_\_$$

Coercions between functions

$$\langle T_1 \to T_2 \Rightarrow U_1 \to U_2 \rangle^{\ell}$$

$$\langle U_1 \Rightarrow T_1 \rangle^{\ell} \qquad\qquad \langle T_2 \Rightarrow U_2 \rangle^{\ell}$$

$$\underline{\qquad\qquad} \;\to\; \underline{\qquad\qquad}$$

Makeup exam

When we execute

$$<(\mathrm{Nat}\rightarrow\mathrm{Nat})\rightarrow\mathrm{Nat}\Rightarrow(\mathrm{Pos}\rightarrow\mathrm{Pos})\rightarrow\mathrm{Pos}>^{\ell}$$

will we check **Nat** or **Pos**
in the domain's domain?

Makeup exam

When we execute

$$\langle (\mathsf{Nat} \to \mathsf{Nat}) \to \mathsf{Nat} \Rightarrow (\mathsf{Pos} \to \mathsf{Pos}) \to \mathsf{Pos} \rangle^{\ell}$$

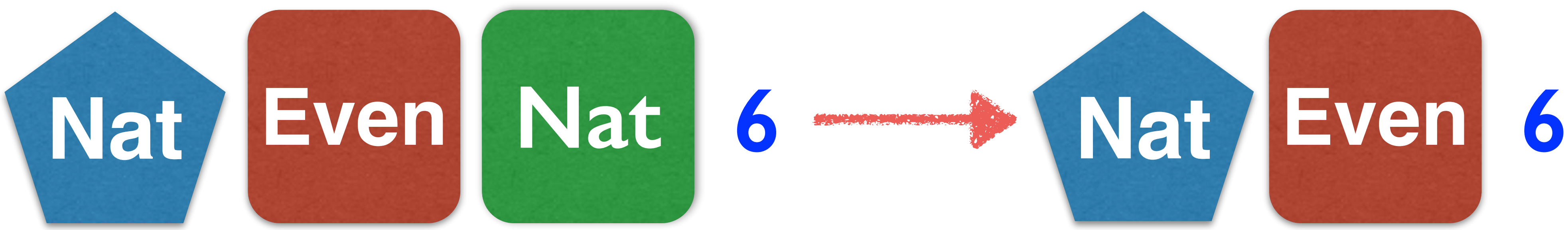
$$(\boxed{\mathsf{Pos}} \to \boxed{\mathsf{Nat}}) \to \boxed{\mathsf{Pos}}$$

will we check **Nat** or **Pos**
in the domain's domain?

Bodies in Urban Spaces
Willi Dorner / Studio 70

Insight #2:
avoid redundant checks

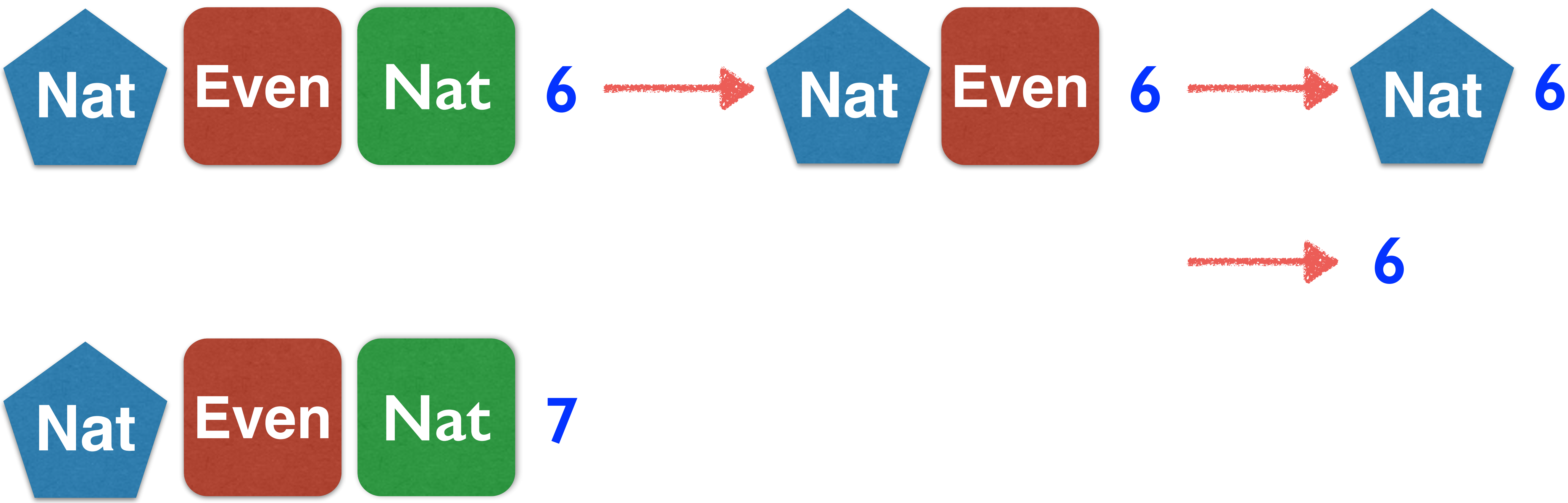
Insight #2:
avoid redundant checks



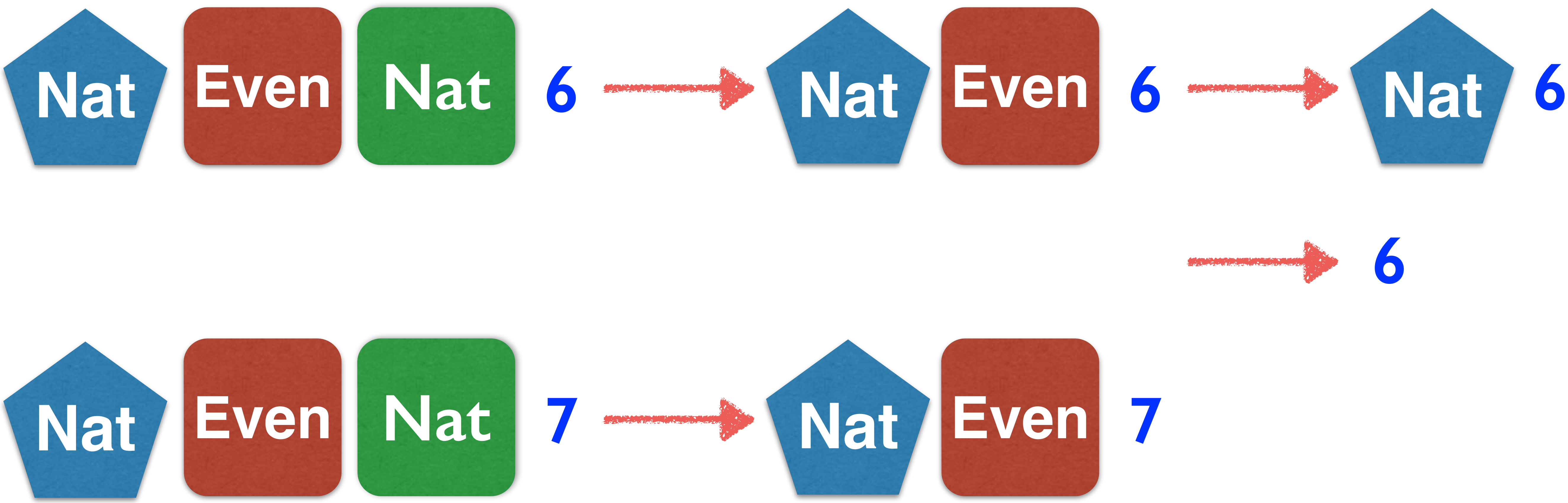
Insight #2:
avoid redundant checks

Insight #2:
avoid redundant checks

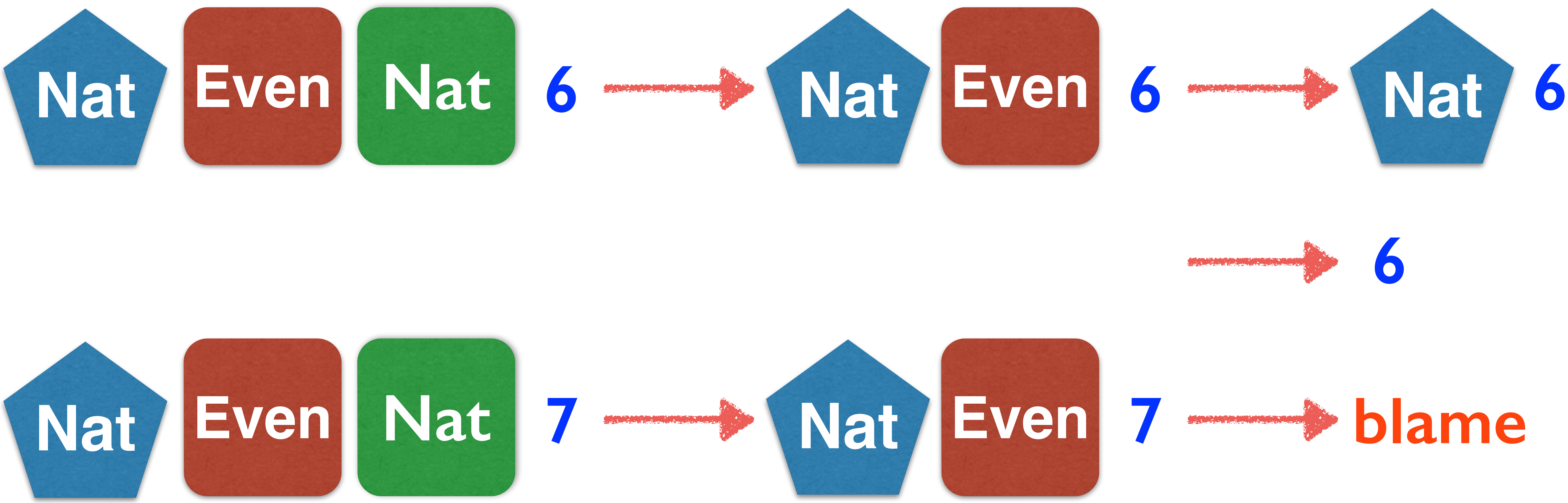
Insight #2:
avoid redundant checks



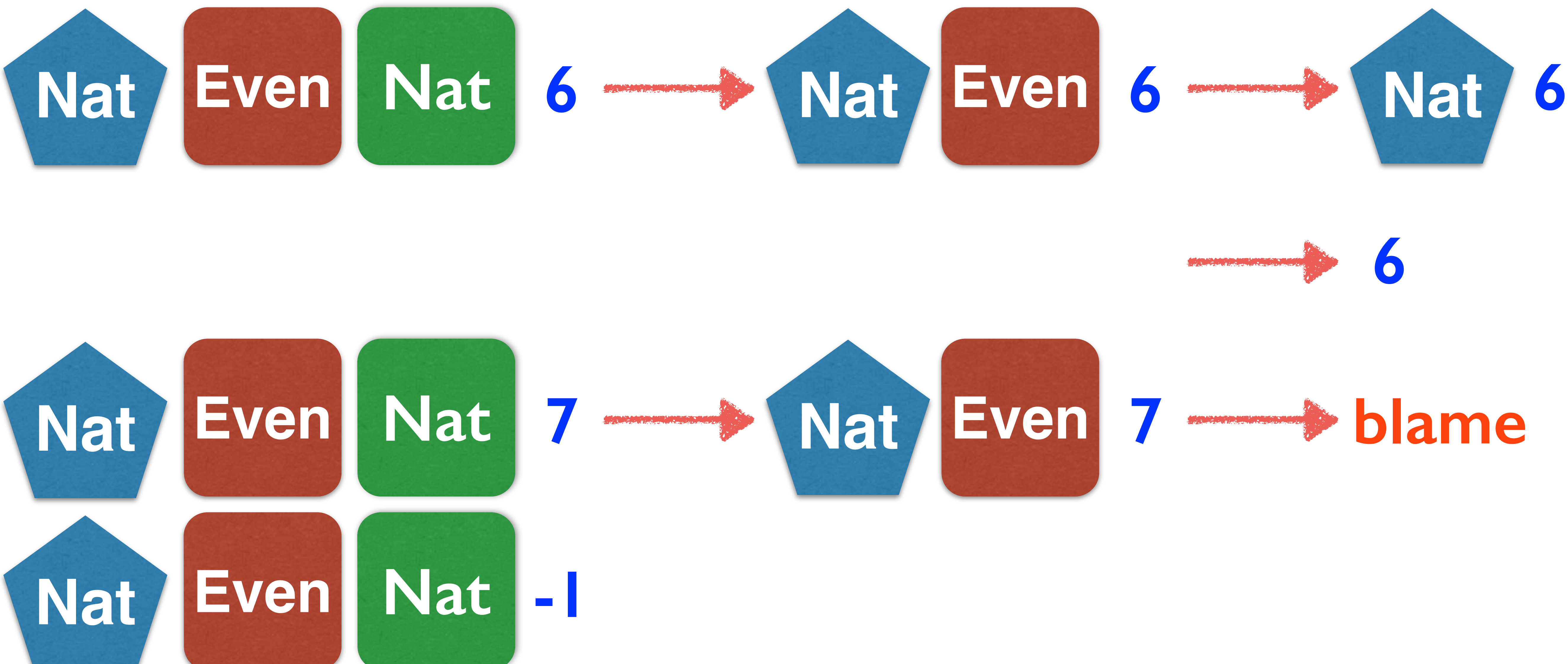
Insight #2:
avoid redundant checks



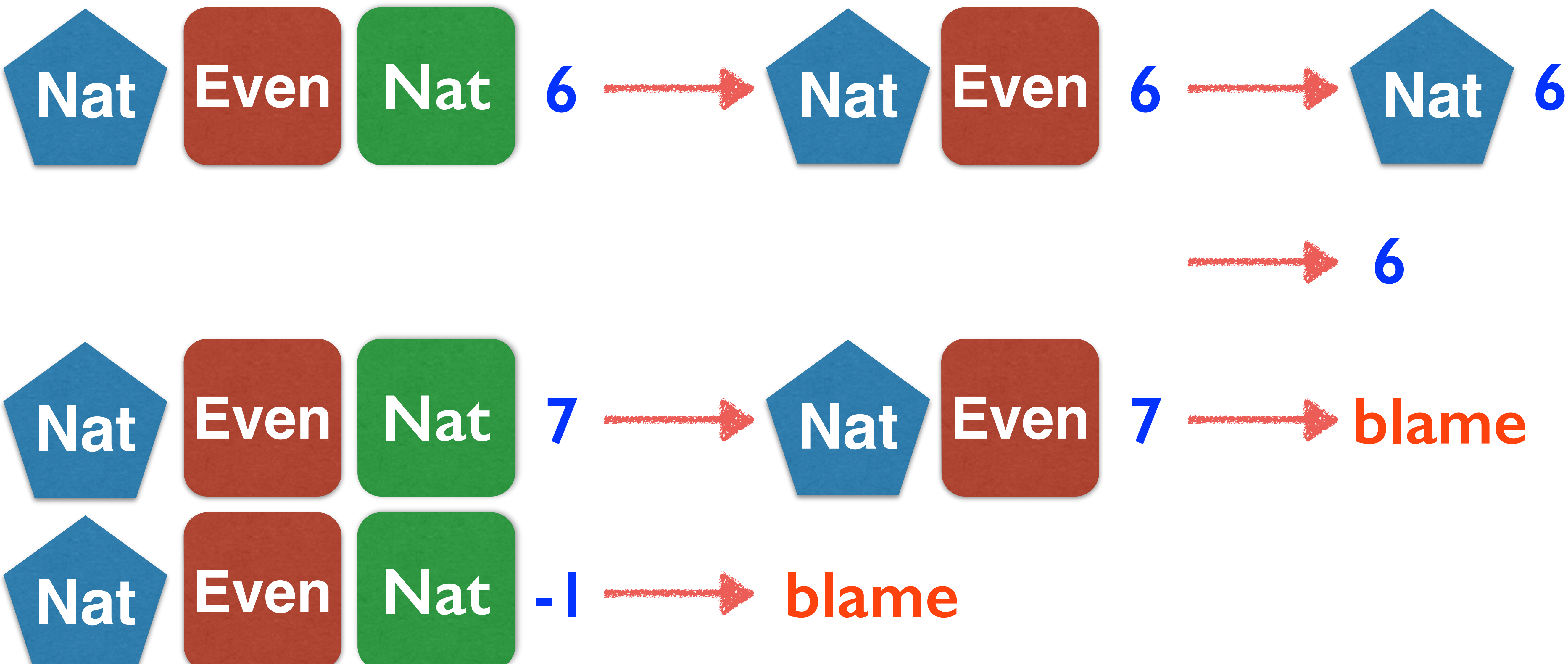
Insight #2: avoid redundant checks



Insight #2: avoid redundant checks

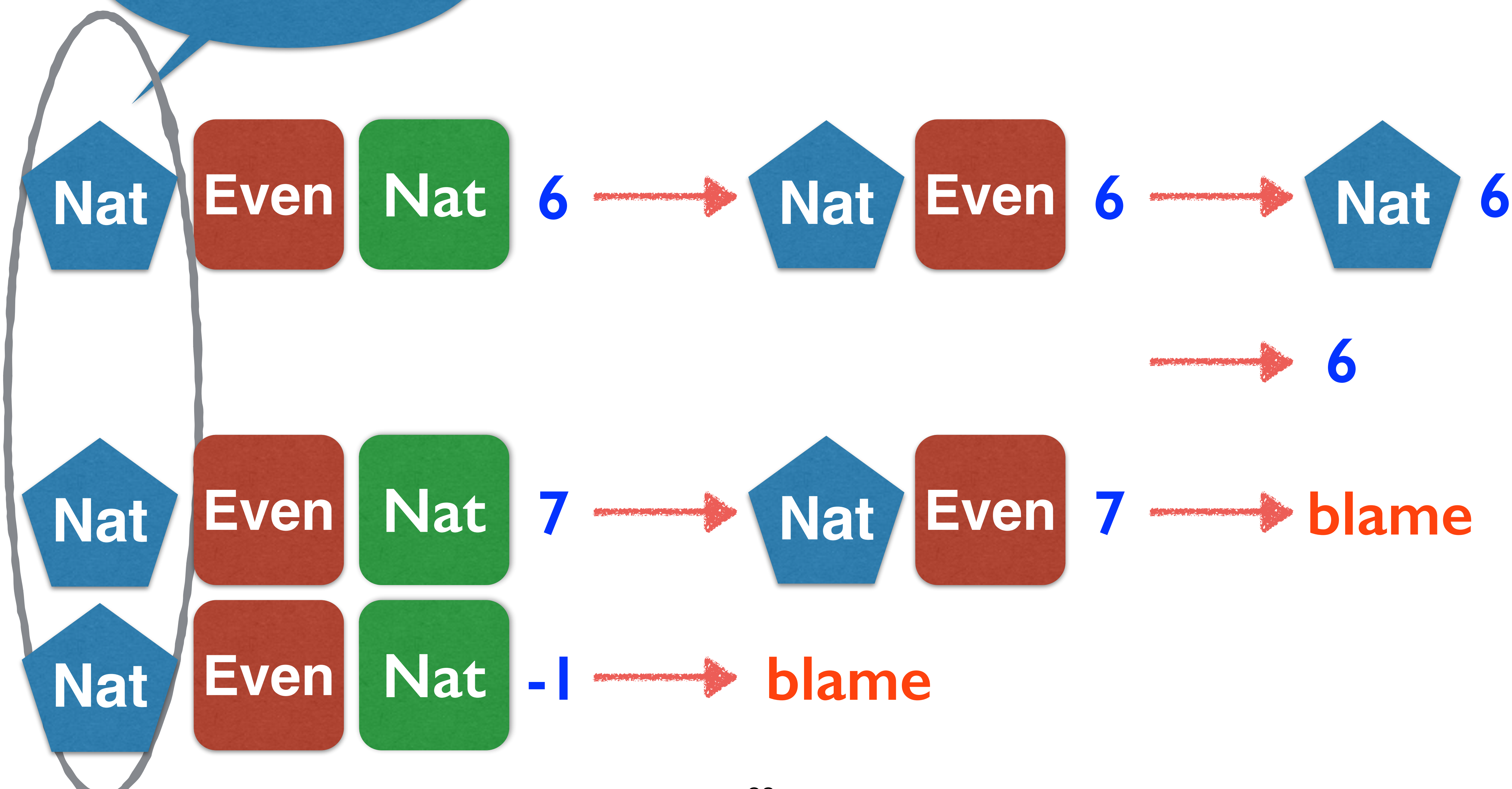


Insight #2:
avoid redundant checks



Insight #2:
redundant checks

Never fails!



Eliminating redundant checks

let odd = → (λn:Int. … even (n-1))

let even = → (λn:Int. … odd (n-1))

even (3)

Eliminating redundant checks

let odd = → (λn:Int. … even (n-1))

let even = → (λn:Int. … odd (n-1))

even (3)

odd (2)

Eliminating redundant checks

let odd = → (λn:Int. … even (n-1))

let even = → (λn:Int. … odd (n-1))

even (3)

odd (2)

even (1)

Eliminating redundant checks

let odd = → (λn:Int. … even (n-1))

let even = → (λn:Int. … odd (n-1))

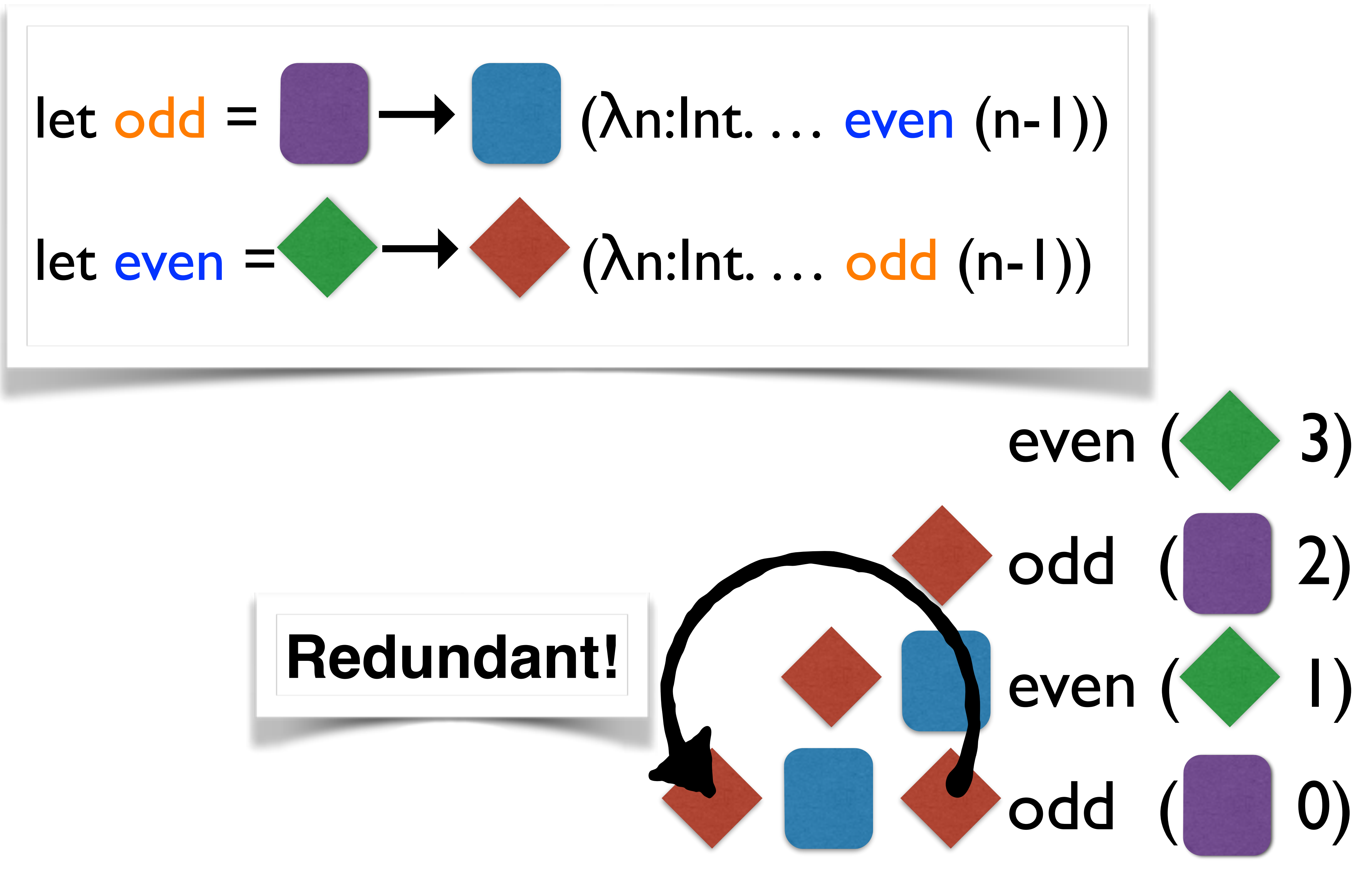
even (3)

odd (2)

even (1)

odd (0)

Eliminating redundant checks



Eliminating redundant checks

let odd = → (λn:Int. … even (n-1))

let even = → (λn:Int. … odd (n-1))

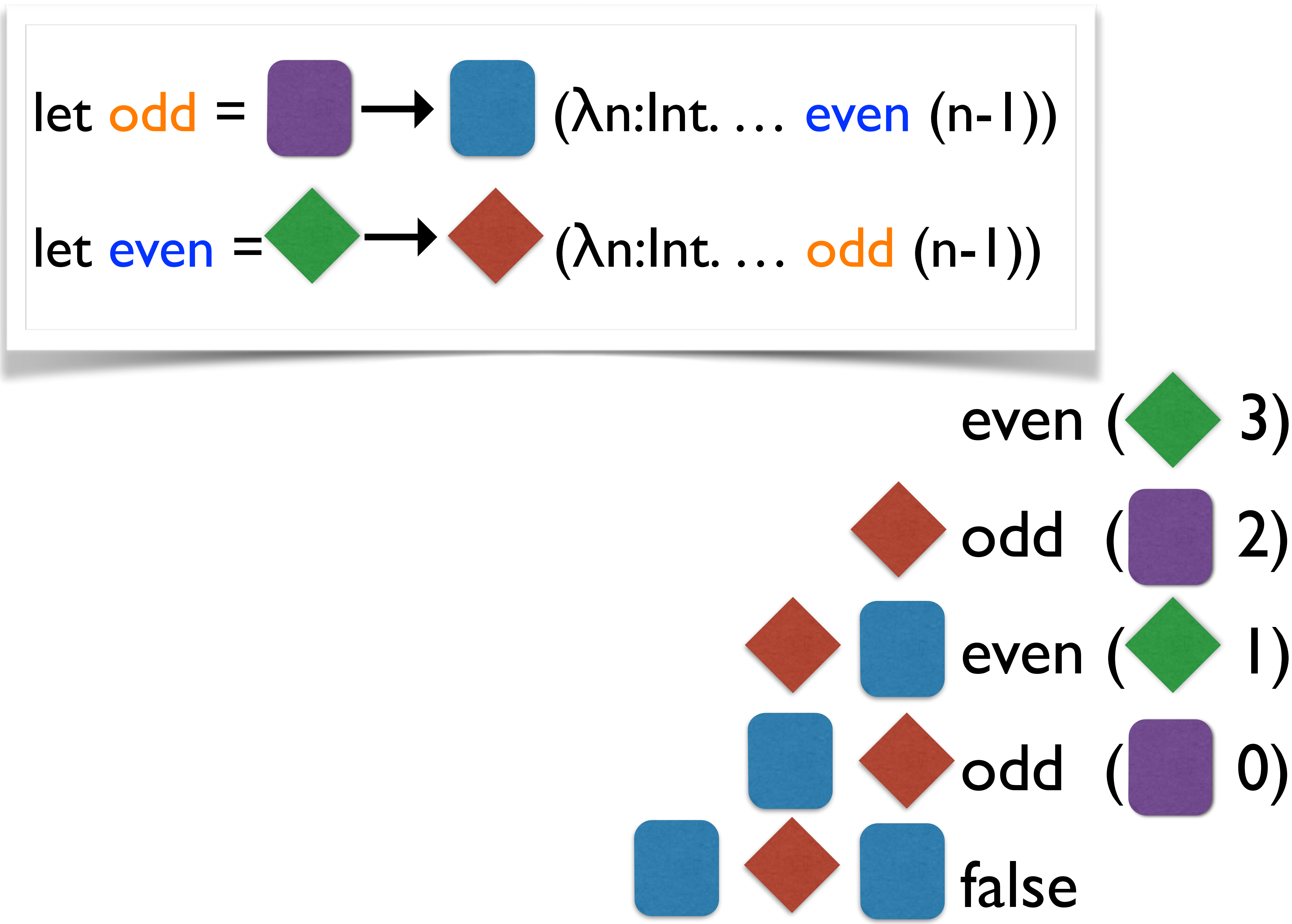
even (3)

odd (2)

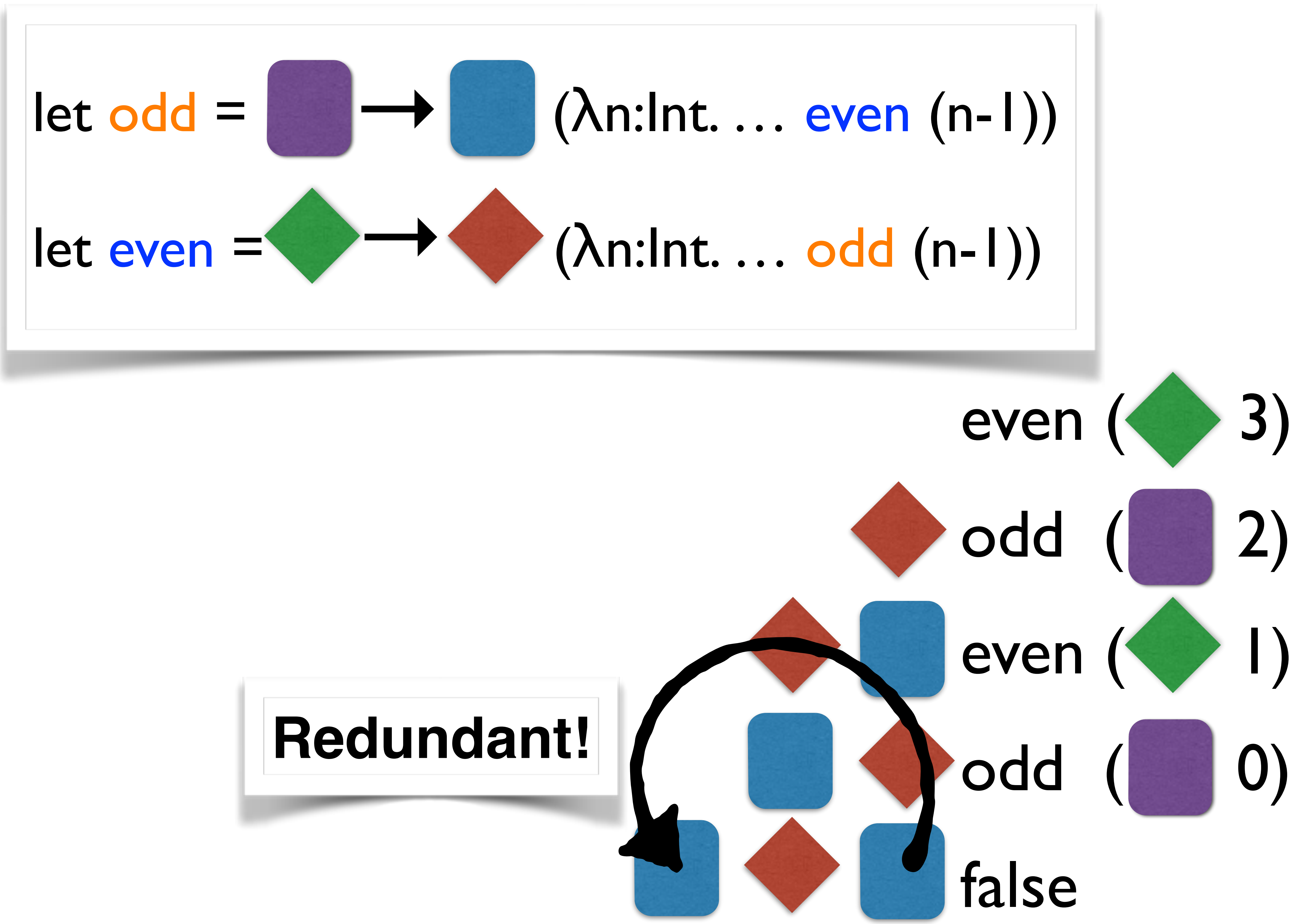
even (1)

odd (0)

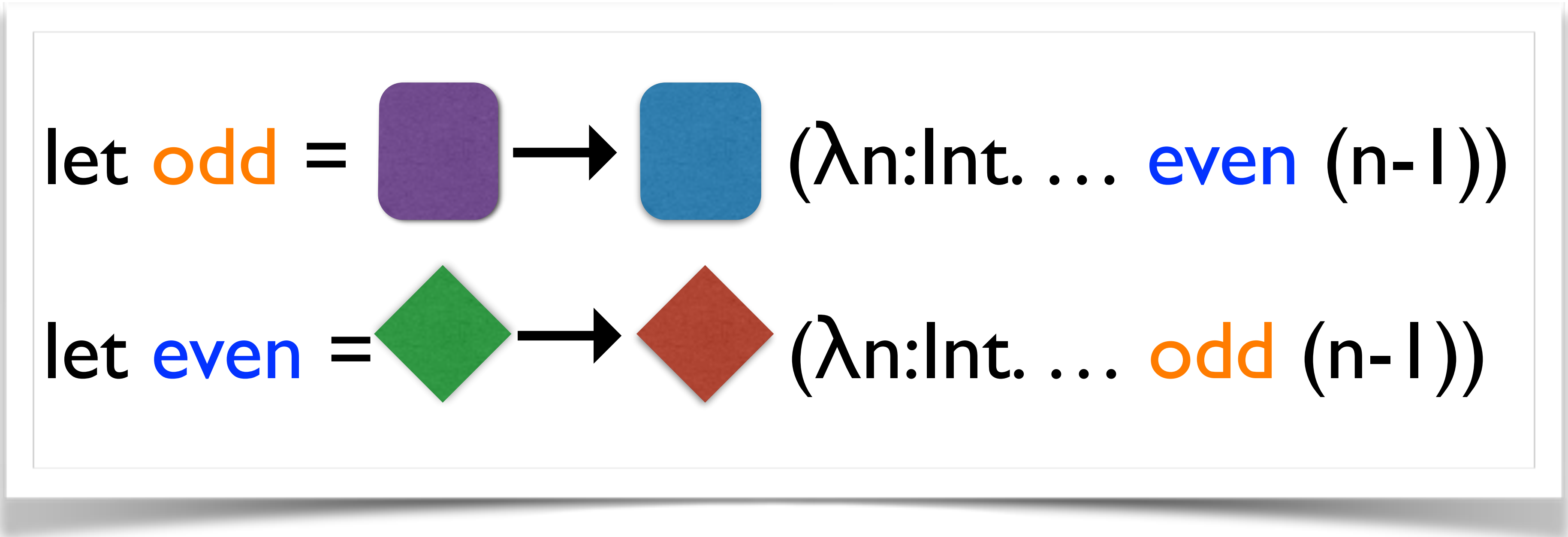
Eliminating redundant checks

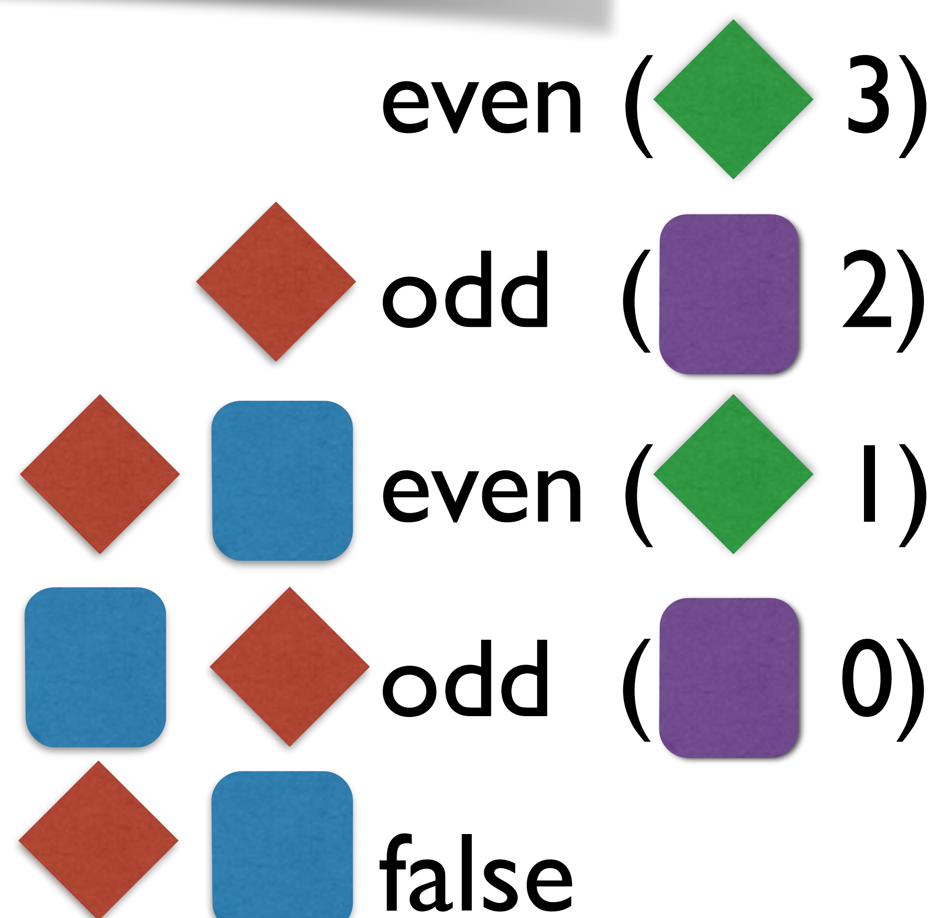


Eliminating redundant checks



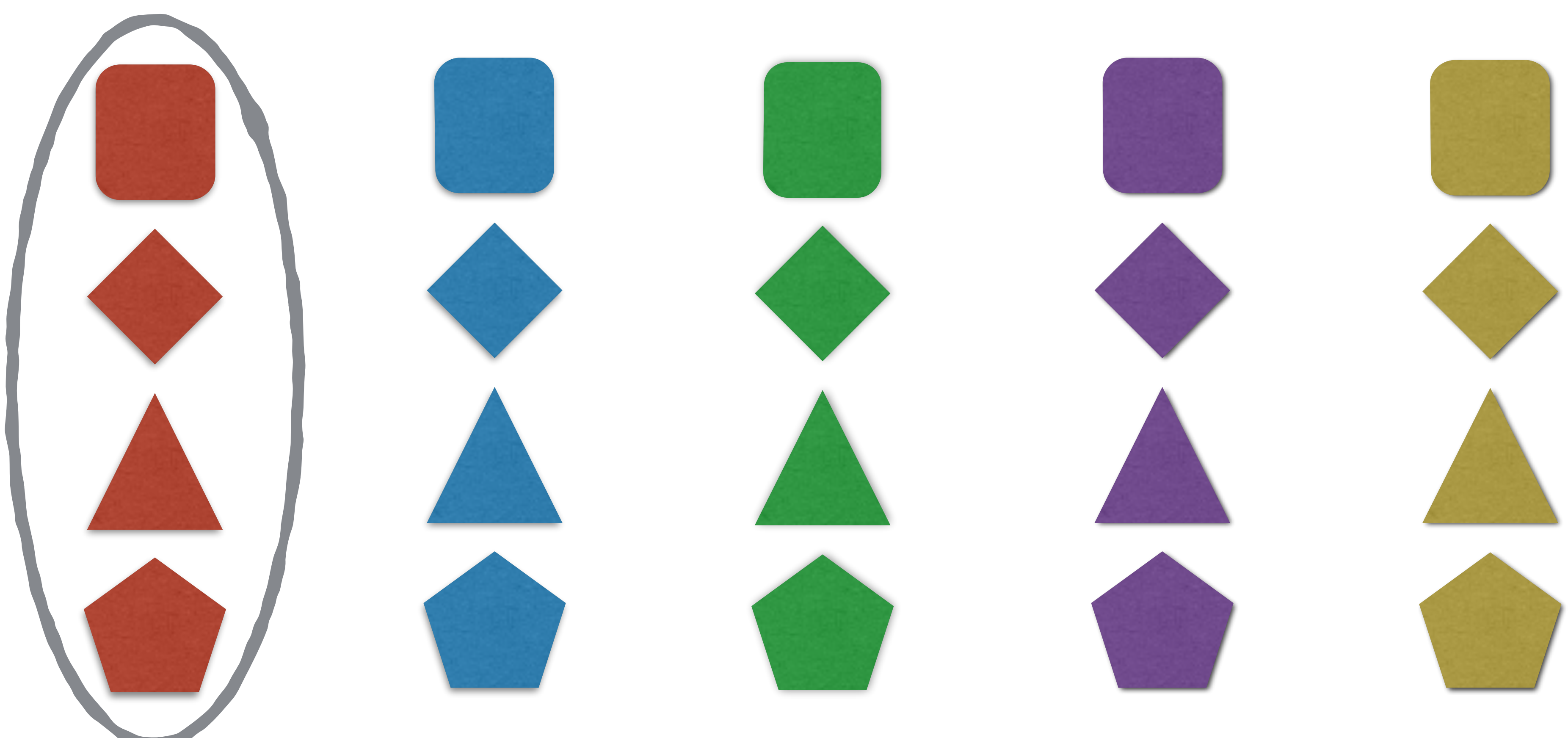
Eliminating redundant checks



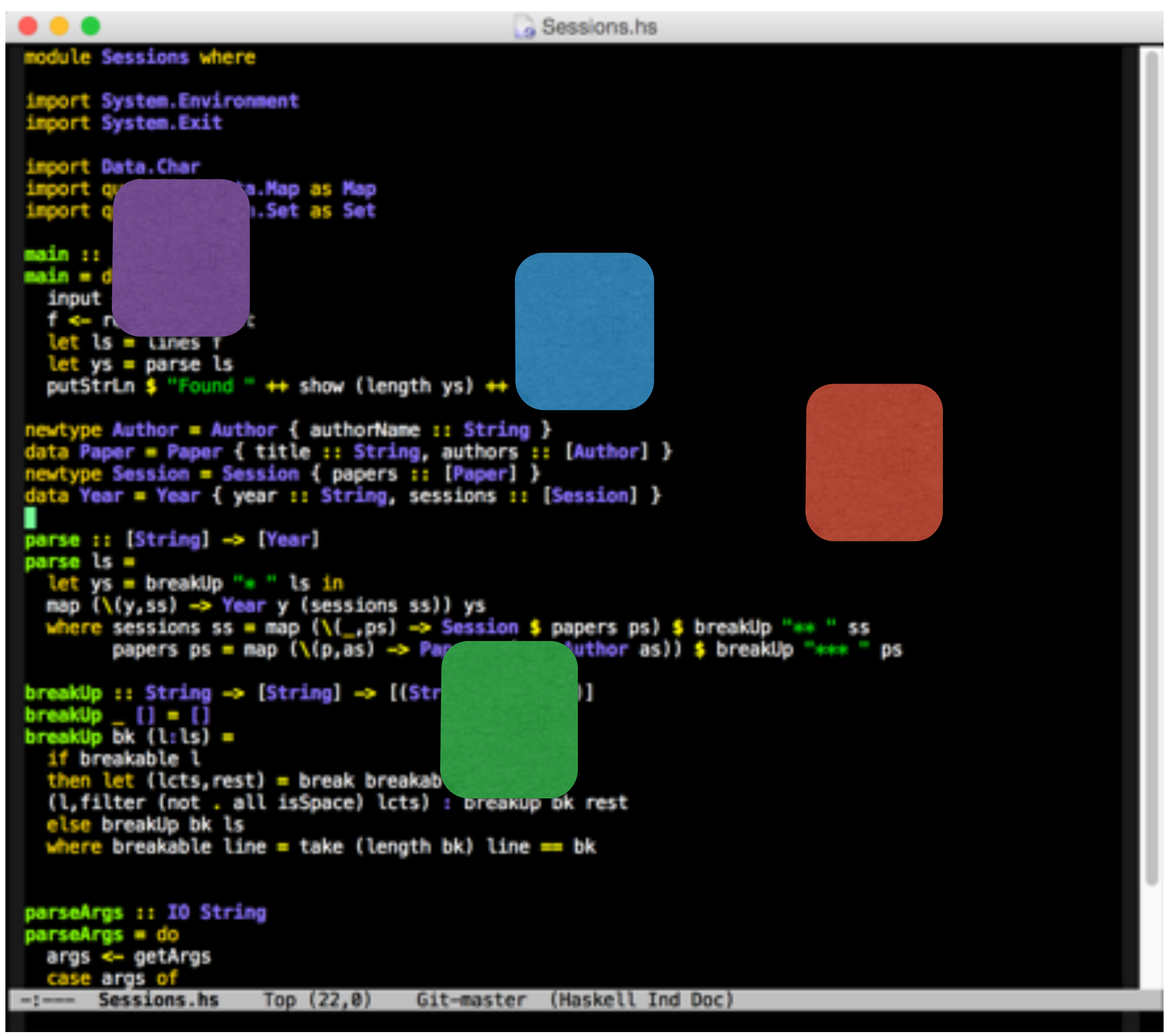


Redundant checks

- Same **color** — same **check**
    - Formally: decidable pre-order on refinement types
- Is this **enough**?



How many checks?

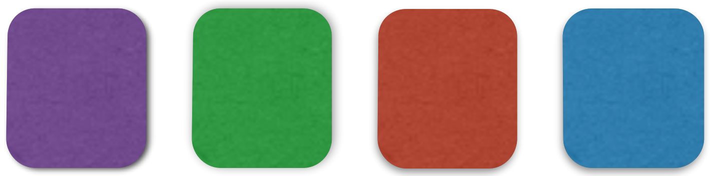


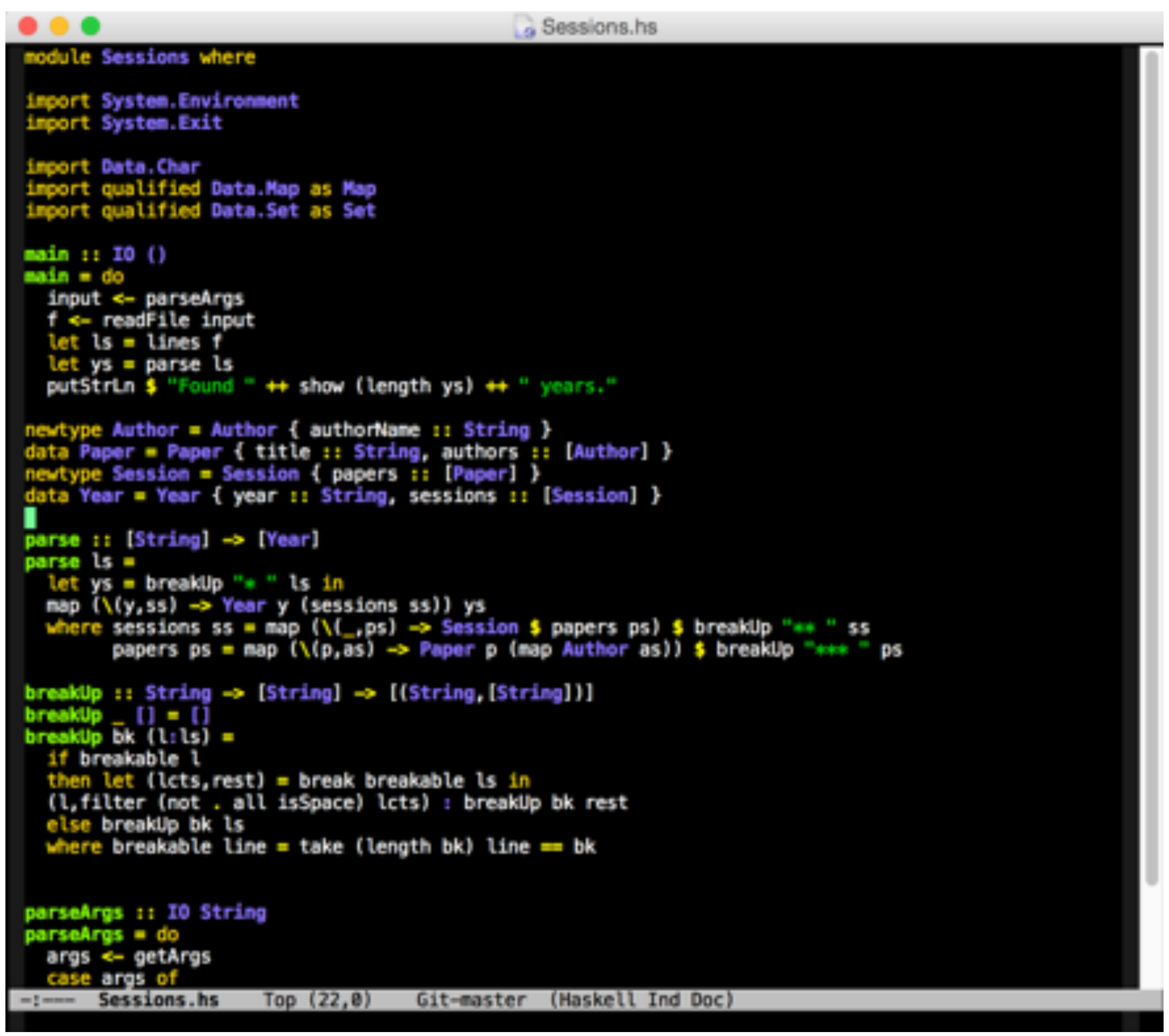
Finitely many

…because of **simple** types!

How many checks?

Types:





Sessions.hs

```haskell
module Sessions where

import System.Environment
import System.Exit

import Data.Char
import qualified Data.Map as Map
import qualified Data.Set as Set

main :: IO ()
main = do
  input <- parseArgs
  f <- readFile input
  let ls = lines f
  let ys = parse ls
  putStrLn $ "Found " ++ show (length ys) ++ " years."

newtype Author = Author { authorName :: String }
data Paper = Paper { title :: String, authors :: [Author] }
newtype Session = Session { papers :: [Paper] }
data Year = Year { year :: String, sessions :: [Session] }

parse :: [String] -> [Year]
parse ls =
  let ys = breakUp "* " ls in
  map (\(y,ss) -> Year y (sessions ss)) ys
  where sessions ss = map (\(_,ps) -> Session $ papers ps) $ breakUp "** " ss
        papers ps = map (\(p,as) -> Paper p (map Author as)) $ breakUp "*** " ps

breakUp :: String -> [String] -> [(String,[String])]
breakUp _ [] = []
breakUp bk (l:ls) =
  if breakable l
  then let (lcts,rest) = break breakable ls in
  (l,filter (not . all isSpace) lcts) : breakUp bk rest
  else breakUp bk ls
  where breakable line = take (length bk) line == bk


parseArgs :: IO String
parseArgs = do
  args <- getArgs
  case args of
-:---  Sessions.hs    Top (22,0)    Git-master  (Haskell Ind Doc)
```

Finitely many

…because of **simple** types!

Bounds

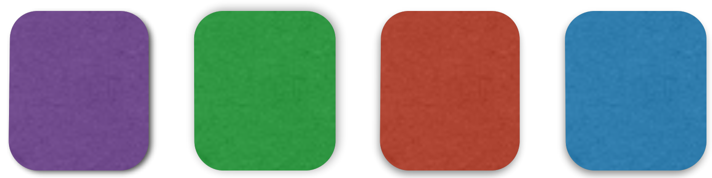
Types:

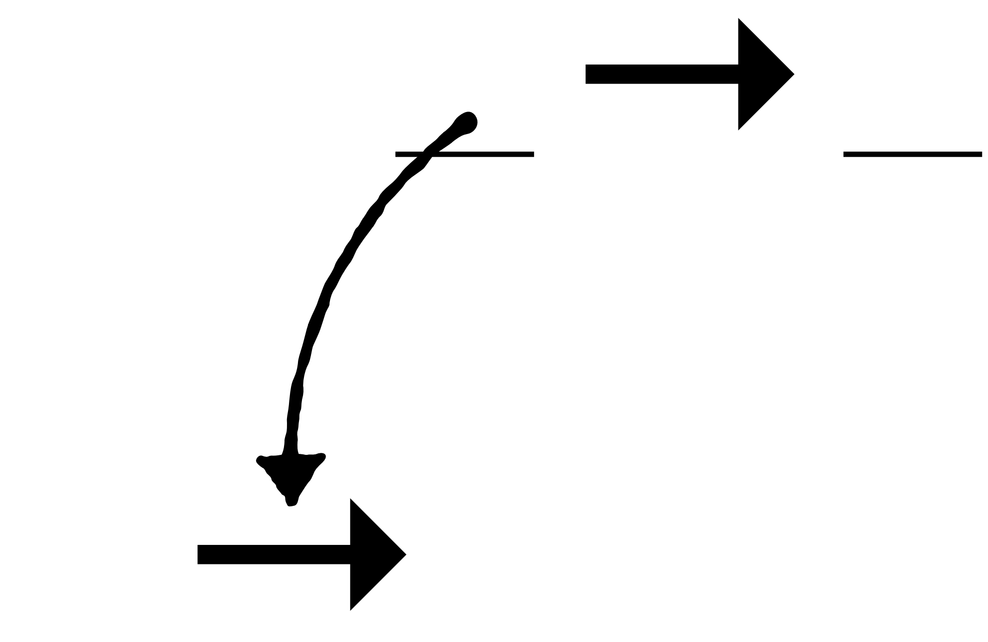
Finitely many types

Appear **once**, at most

What’s the worst that can happen?

Bounds

Types: 



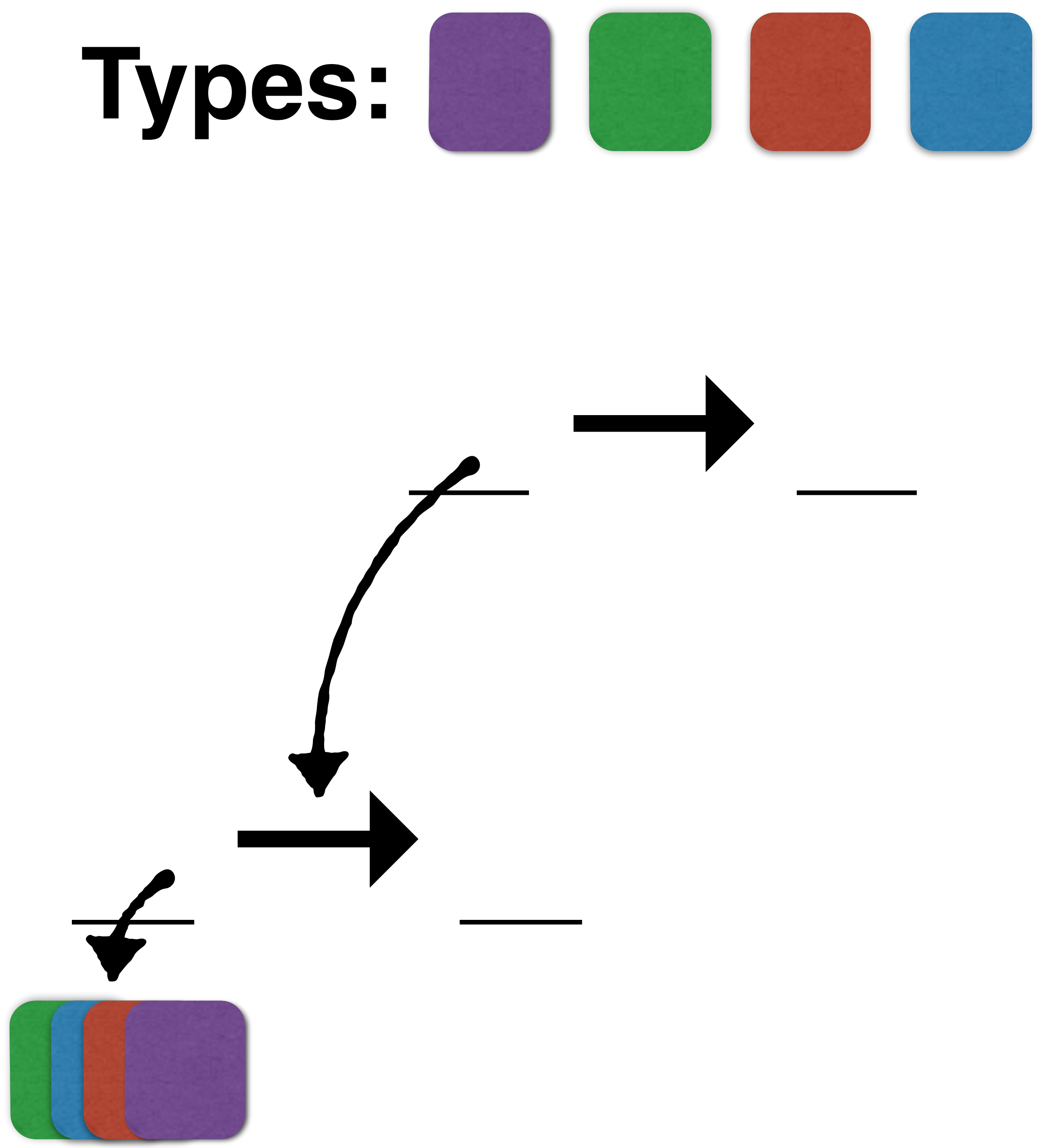
Finitely many types

Appear **once**, at most

What’s the worst that can happen?

Bounds

Types:

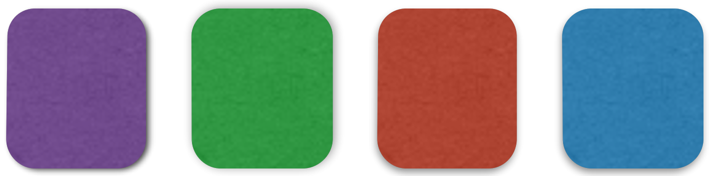


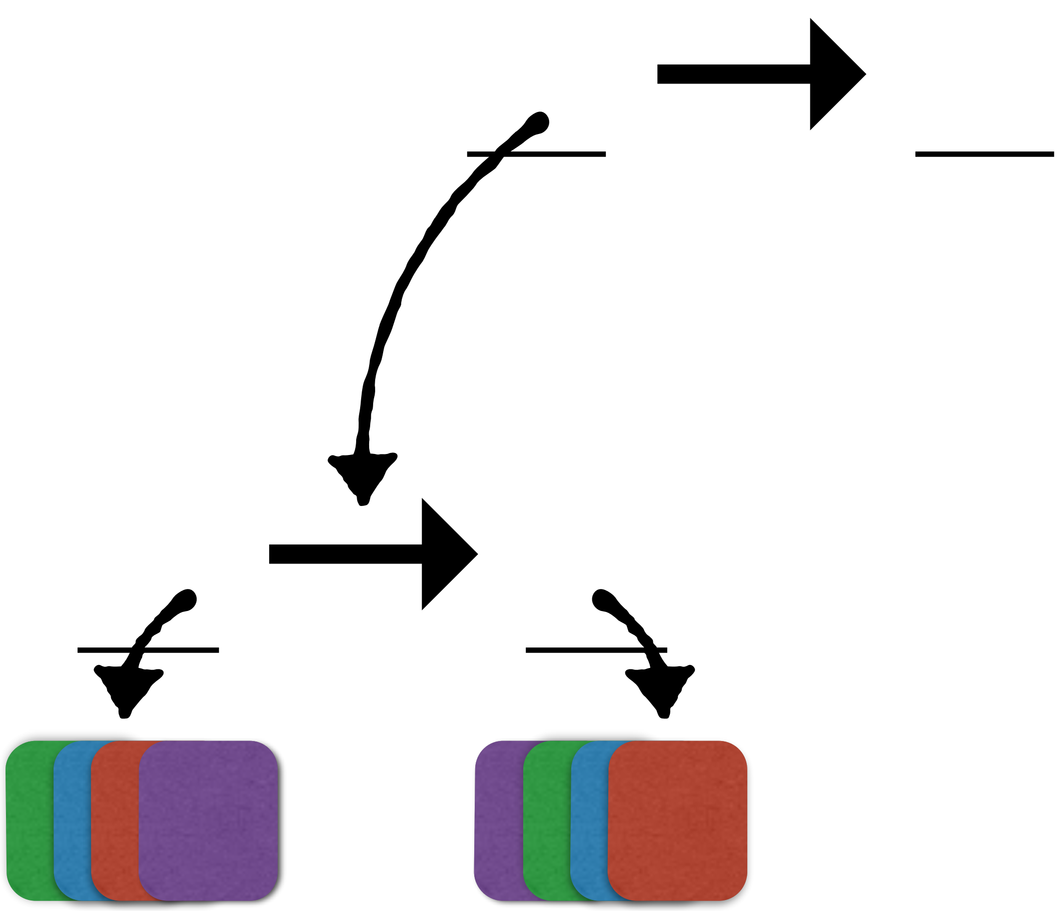
Finitely many types

Appear **once**, at most

What's the worst that can happen?

Bounds

Types: 

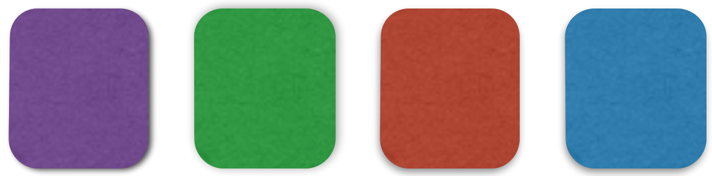


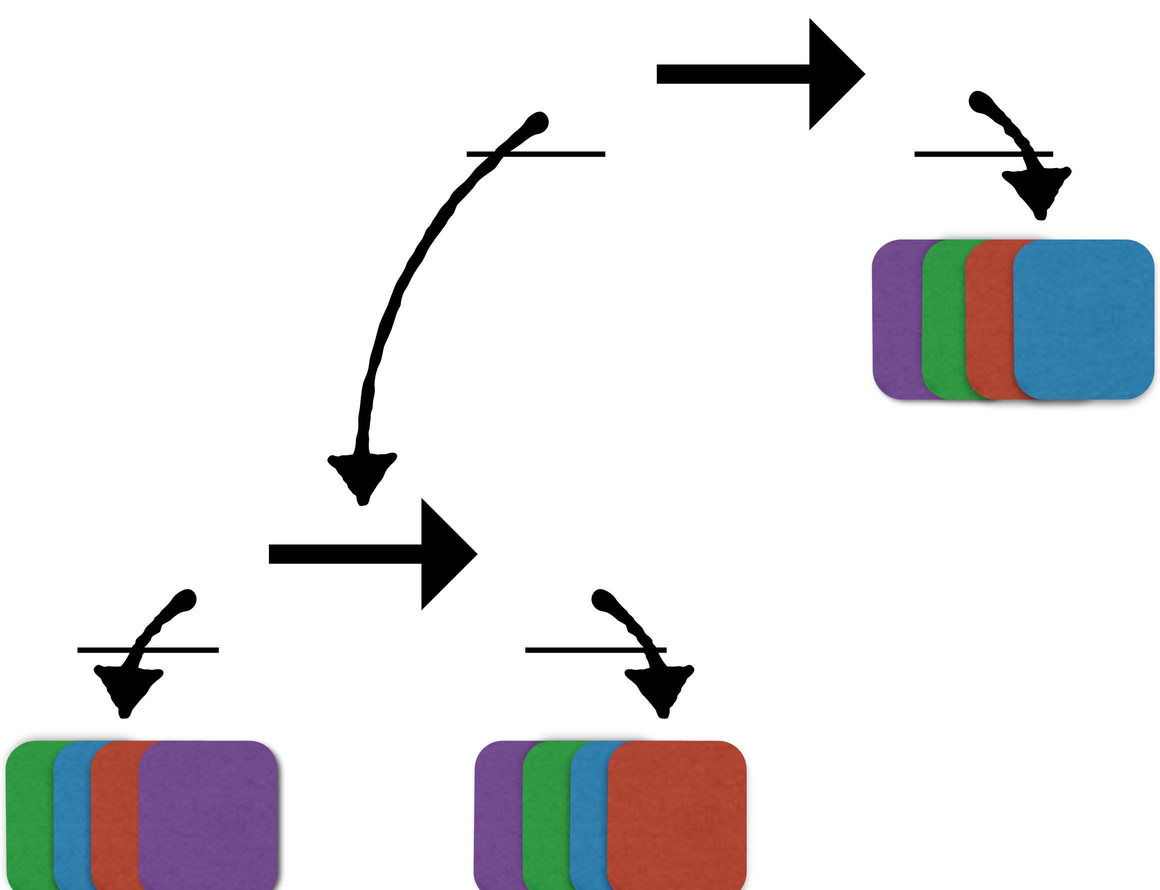
Finitely many types

Appear **once**, at most

What's the worst that can happen?

Bounds

Types: 



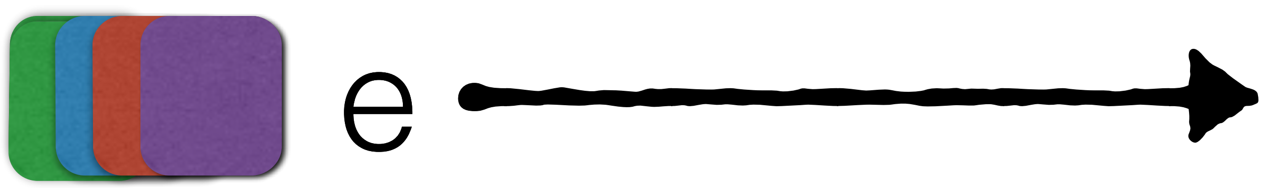
Finitely many types

Appear **once**, at most

What’s the worst that can happen?

Eliminating redundant checks

How do we **merge** lists of checks?

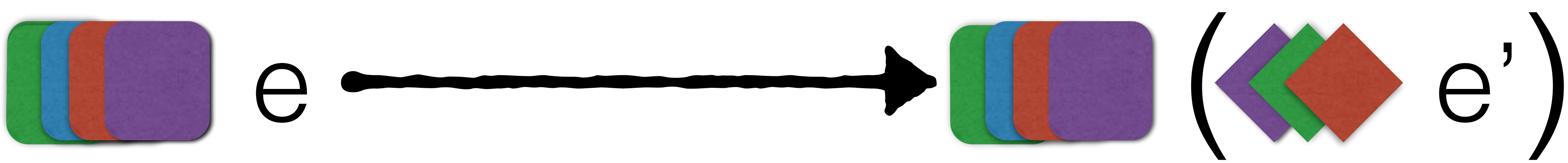


Invariant: the checks on the stack have no **redundancy**.

We'll **merge** the new checks in,
dropping **redundant** checks.

Eliminating redundant checks

How do we **merge** lists of checks?

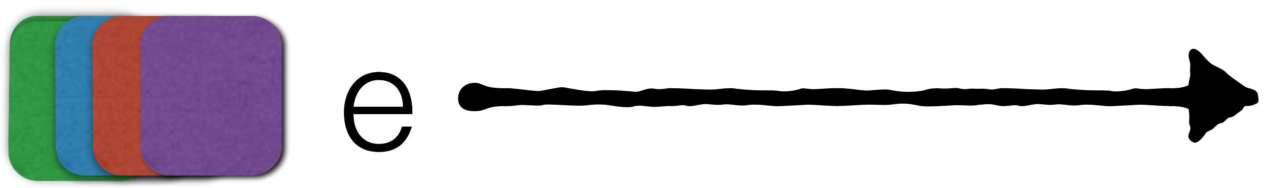


Invariant: the checks on the stack have no **redundancy**.

We’ll **merge** the new checks in,
dropping **redundant** checks.

Merging, in detail

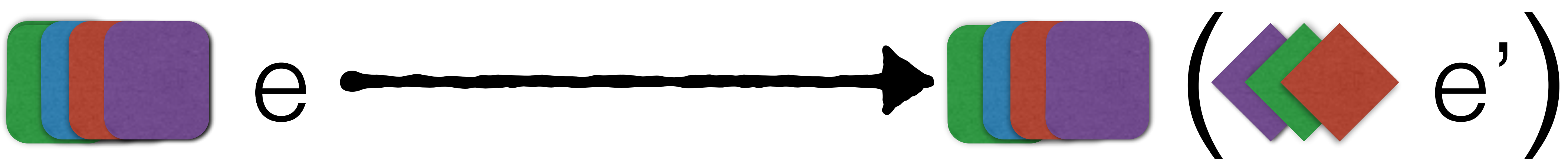
How do we **merge** lists of checks?



\+ = ?

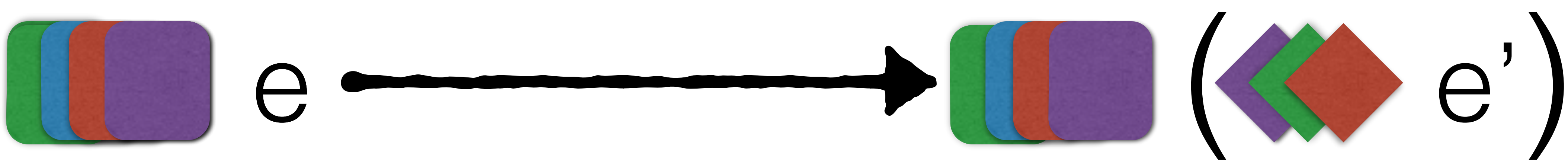
Merging, in detail

How do we **merge** lists of checks?



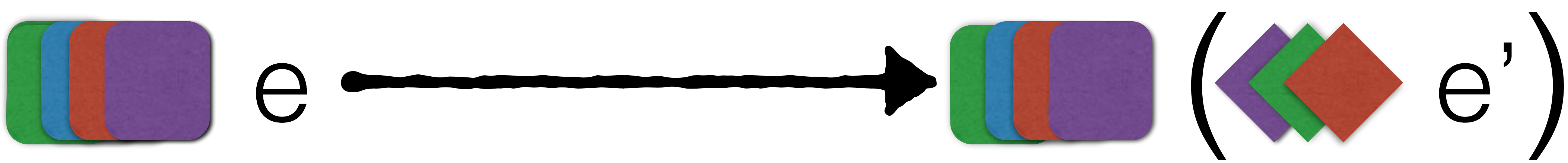
Merging, in detail

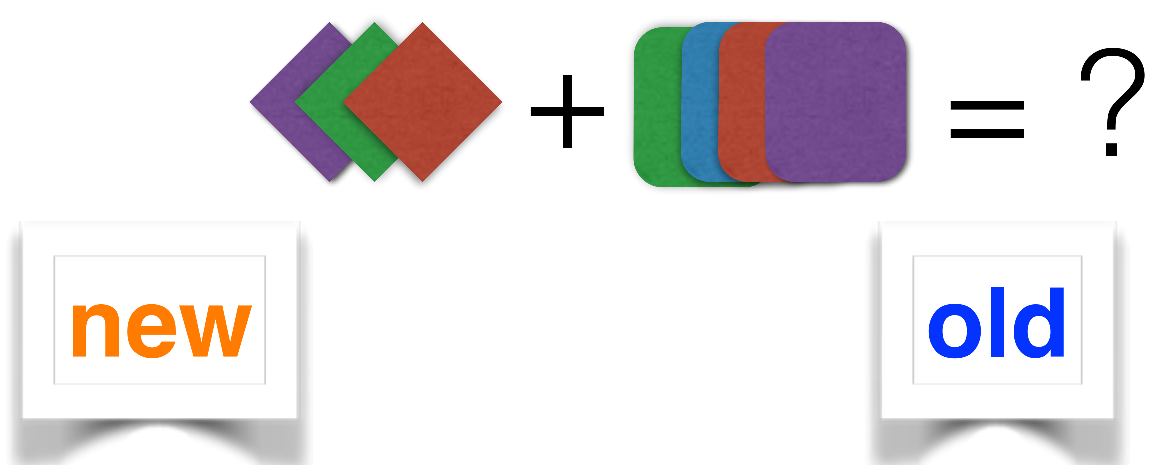
How do we **merge** lists of checks?



Merging, in detail

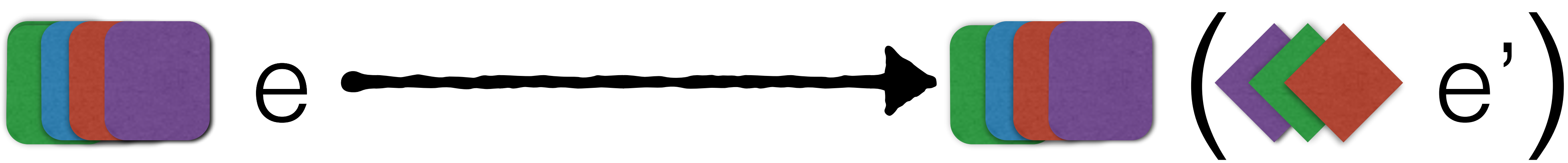
How do we **merge** lists of checks?





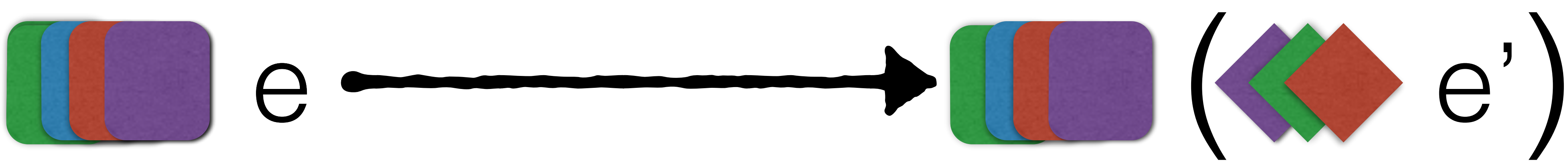
Merging, in detail

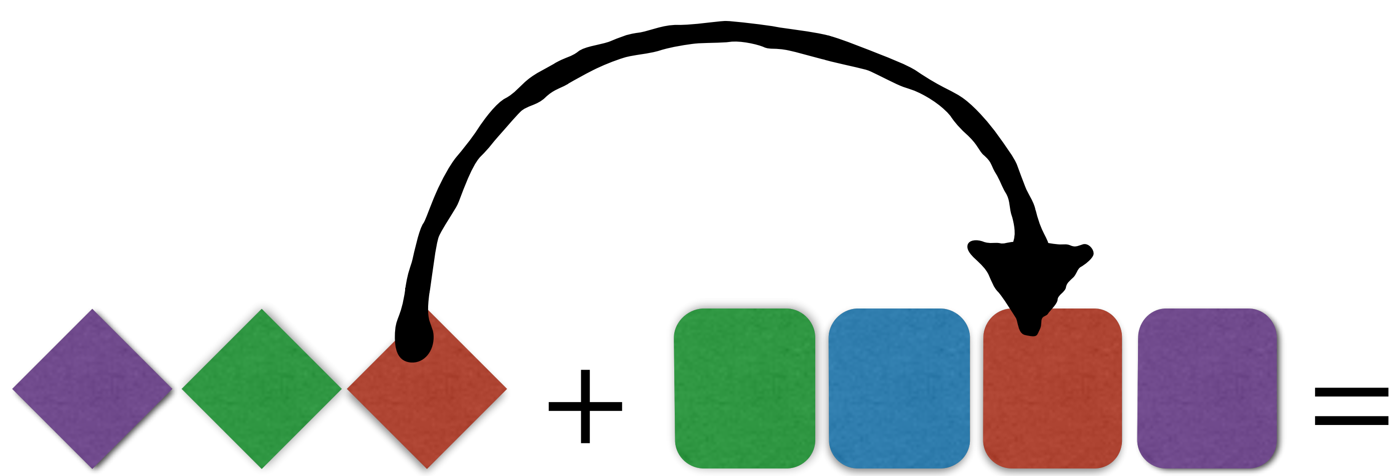
How do we **merge** lists of checks?



Merging, in detail

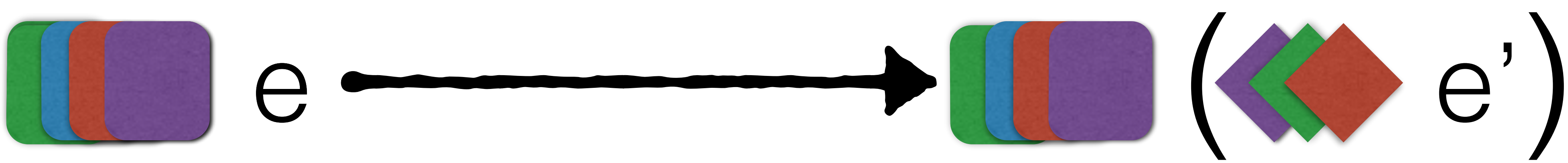
How do we **merge** lists of checks?





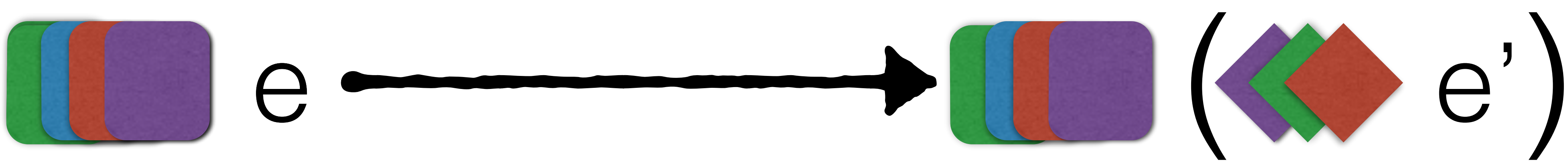
Merging, in detail

How do we **merge** lists of checks?



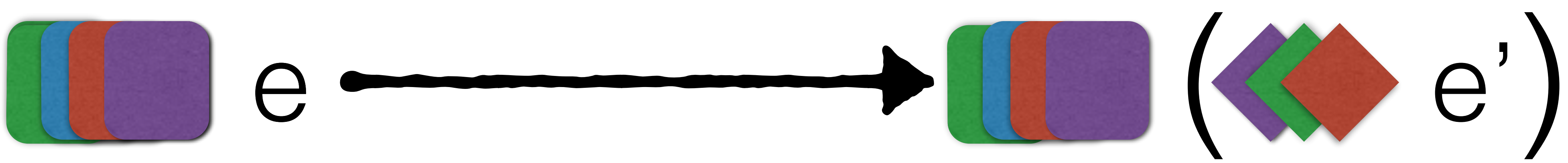
Merging, in detail

How do we **merge** lists of checks?



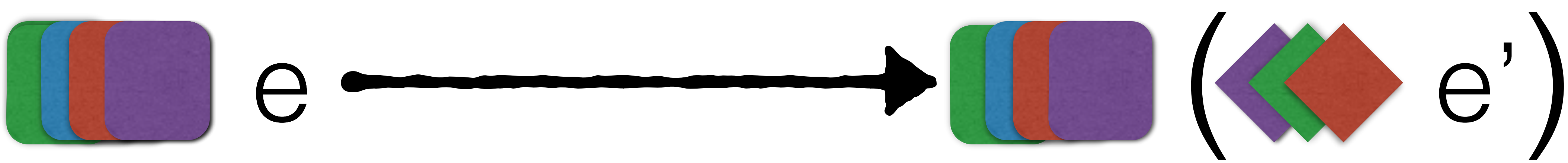
Merging, in detail

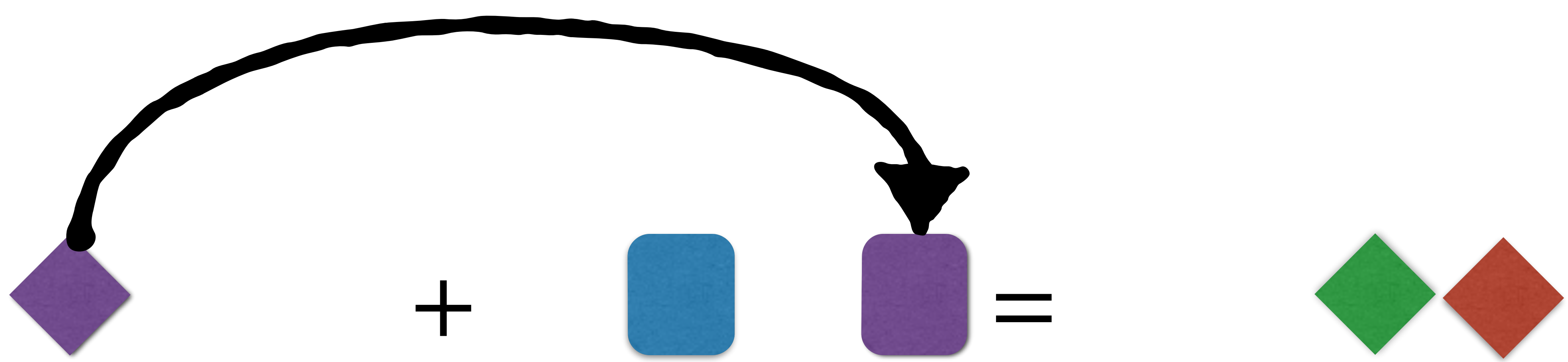
How do we **merge** lists of checks?



Merging, in detail

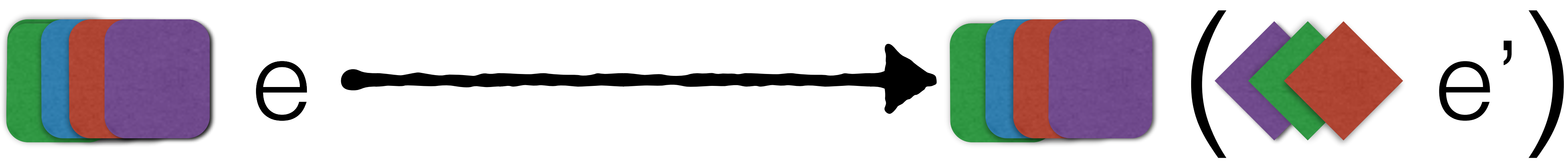
How do we **merge** lists of checks?

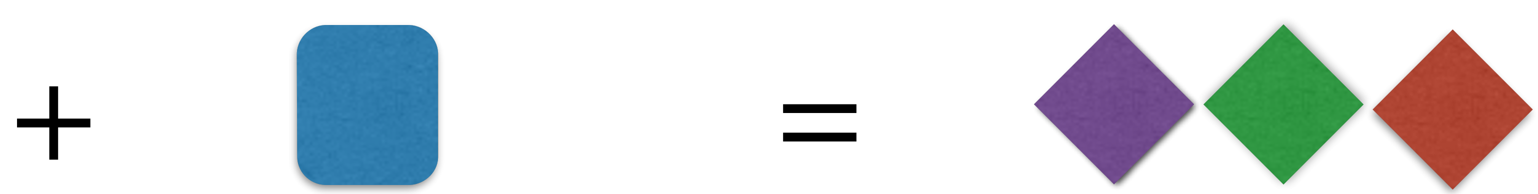




Merging, in detail

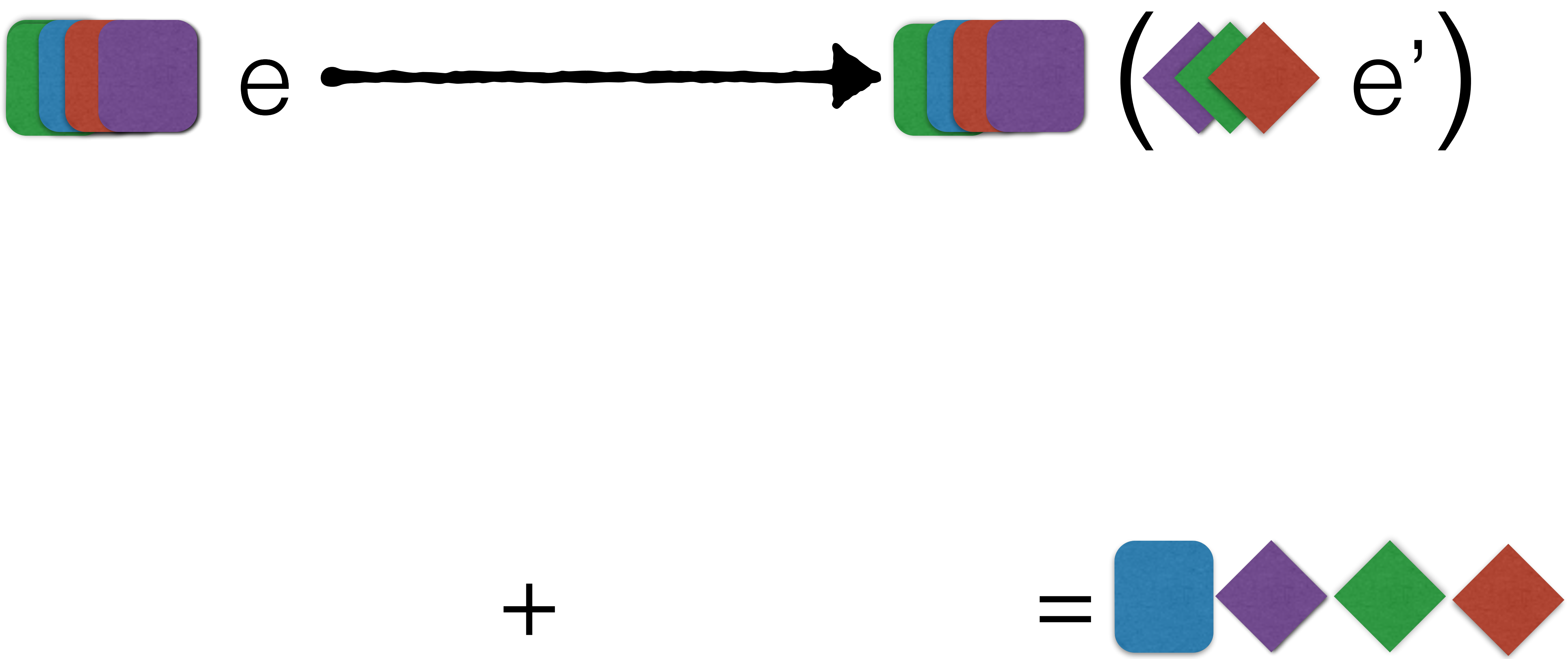
How do we **merge** lists of checks?





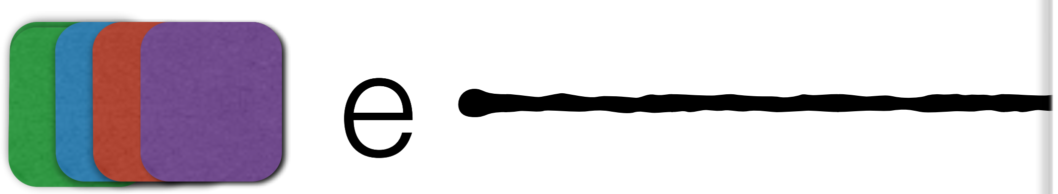
Merging, in detail

How do we **merge** lists of checks?



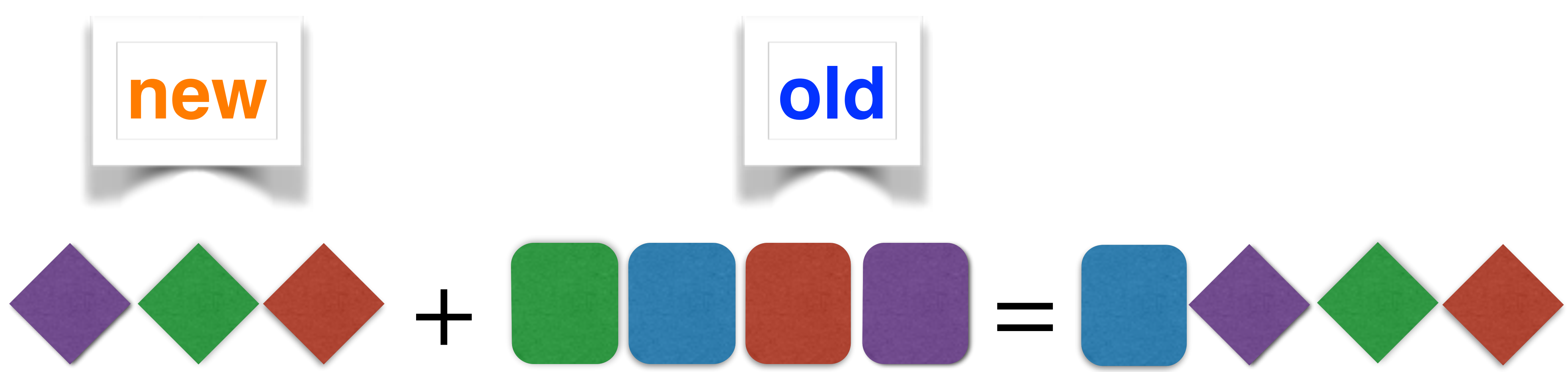
Merging, in detail

How do we **merge** lists o

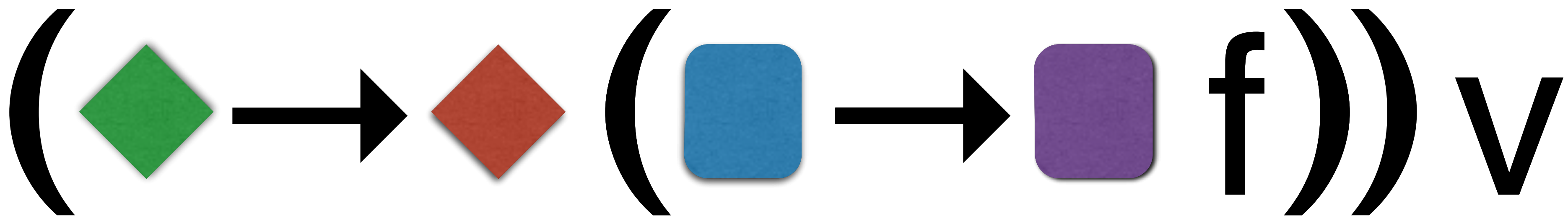


Go from **new** to **old**

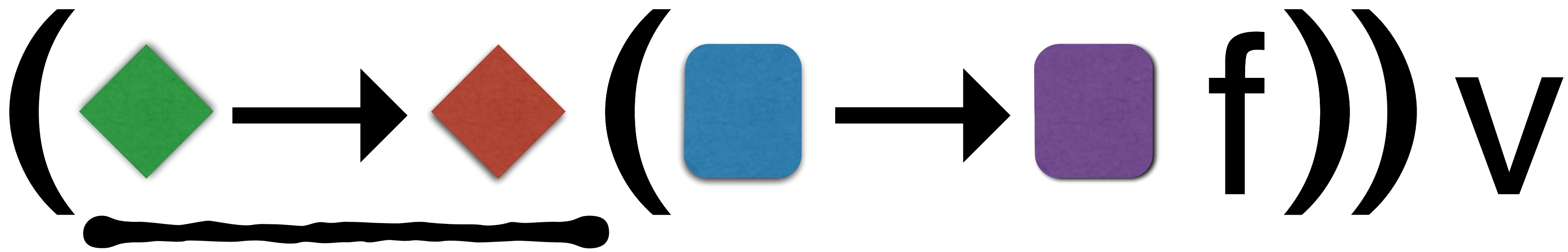
Drop **redundant** checks on the old coercion



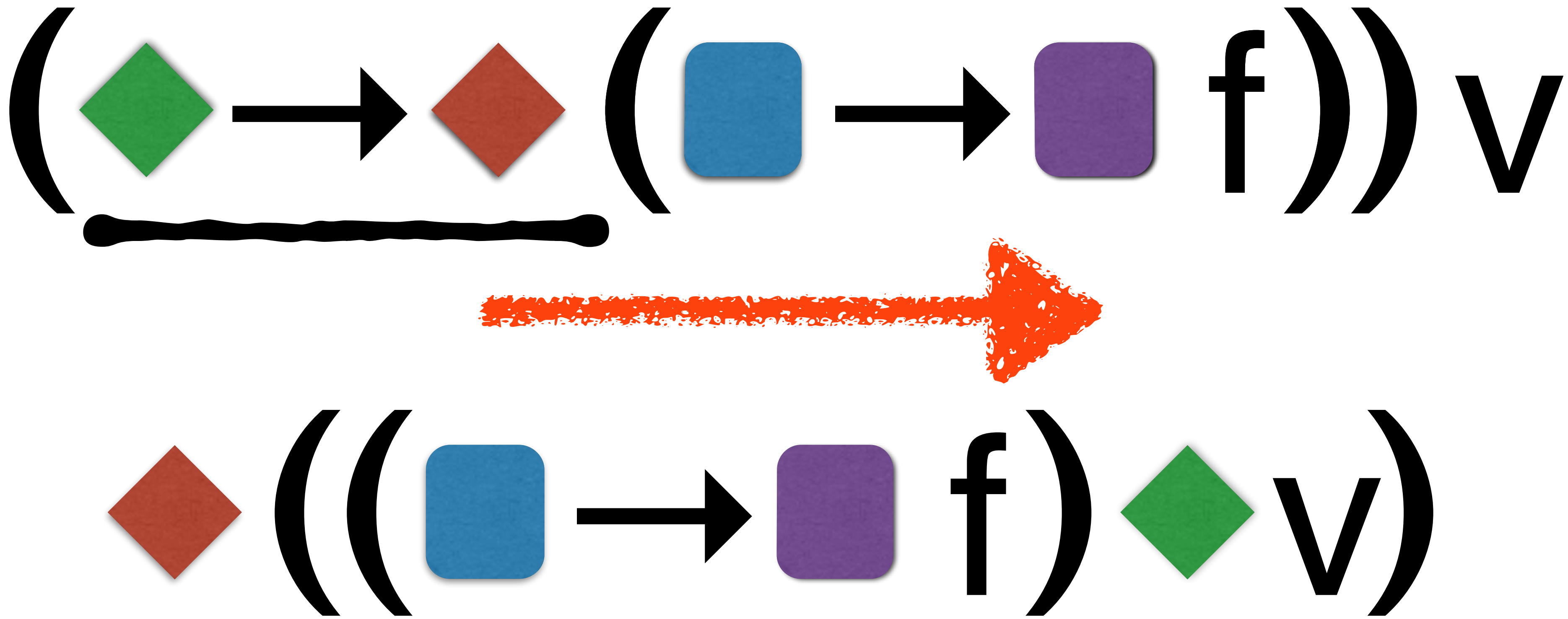
Merging function proxies



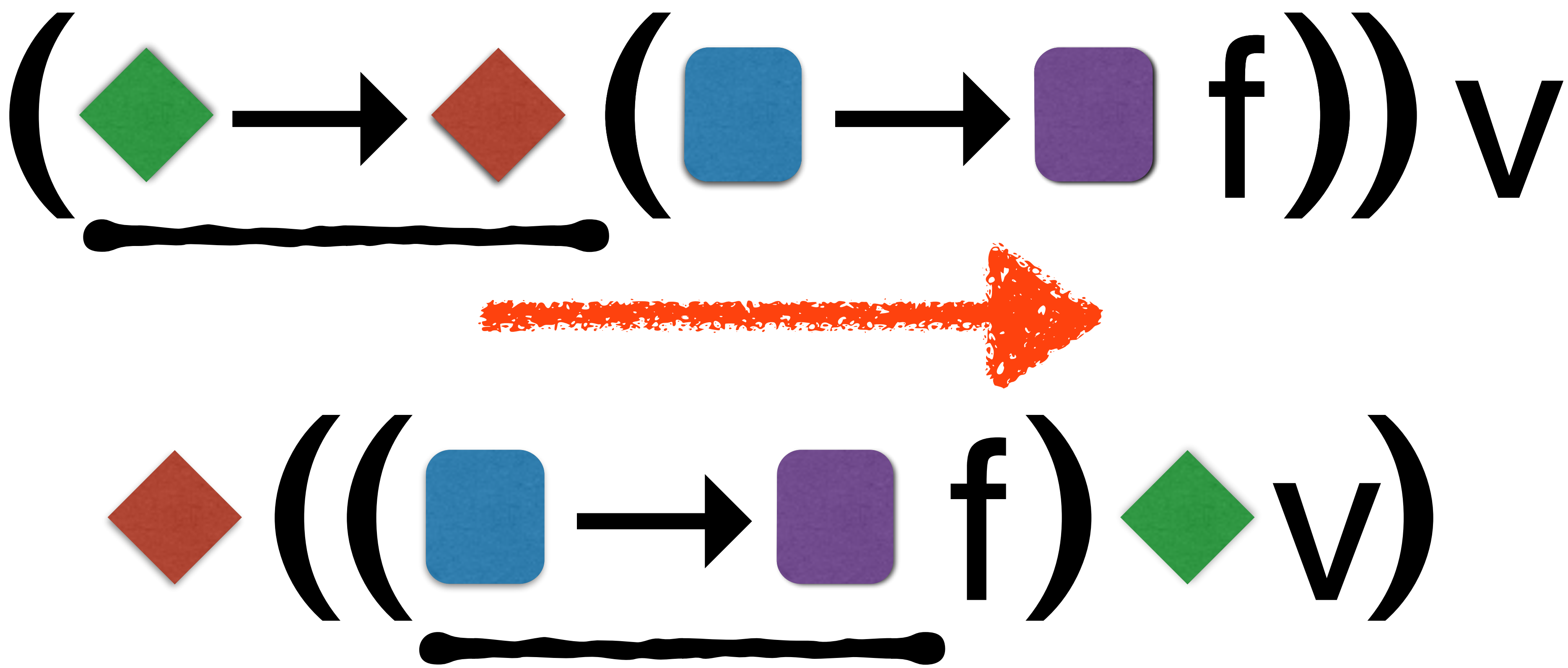
Merging function proxies



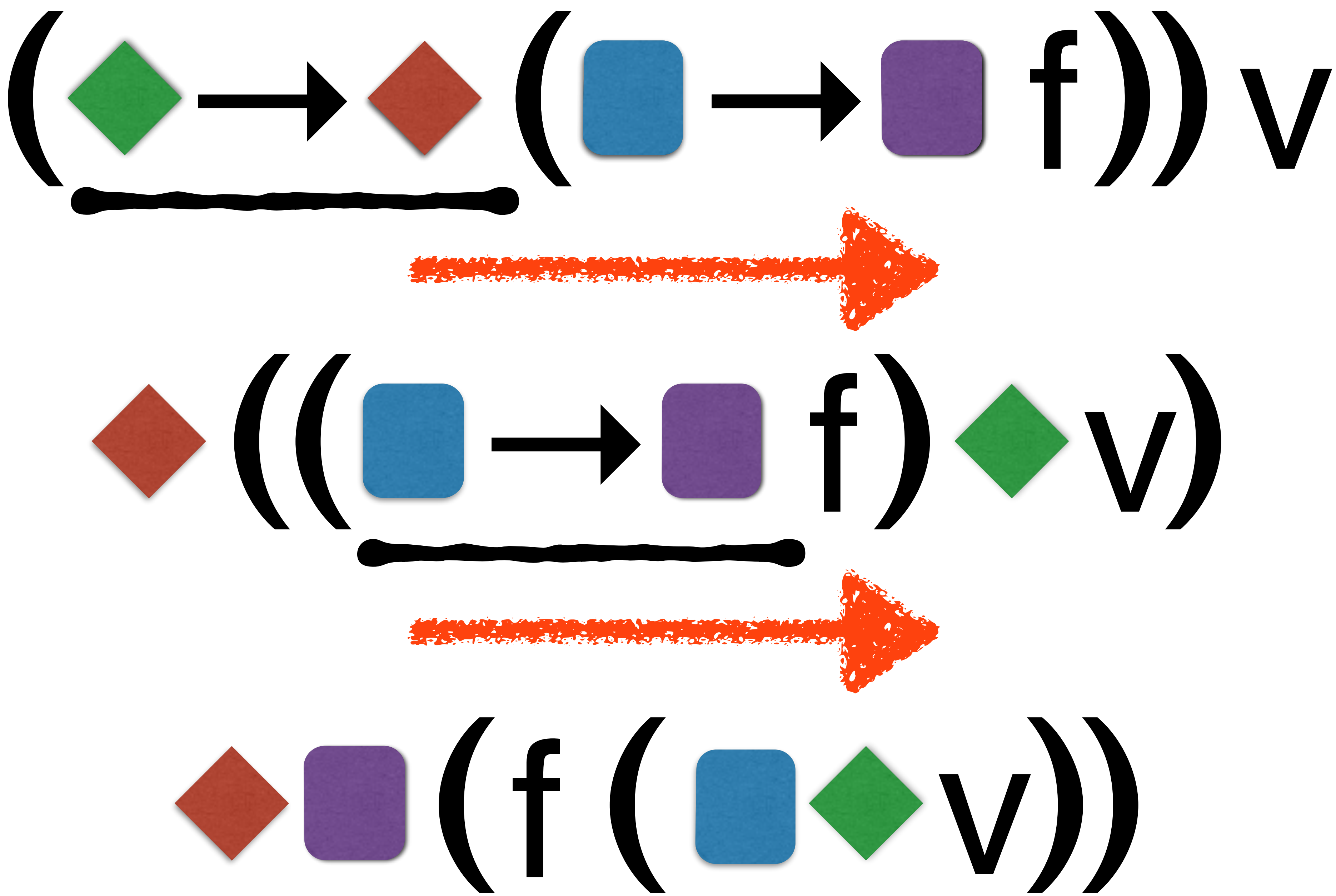
Merging function proxies



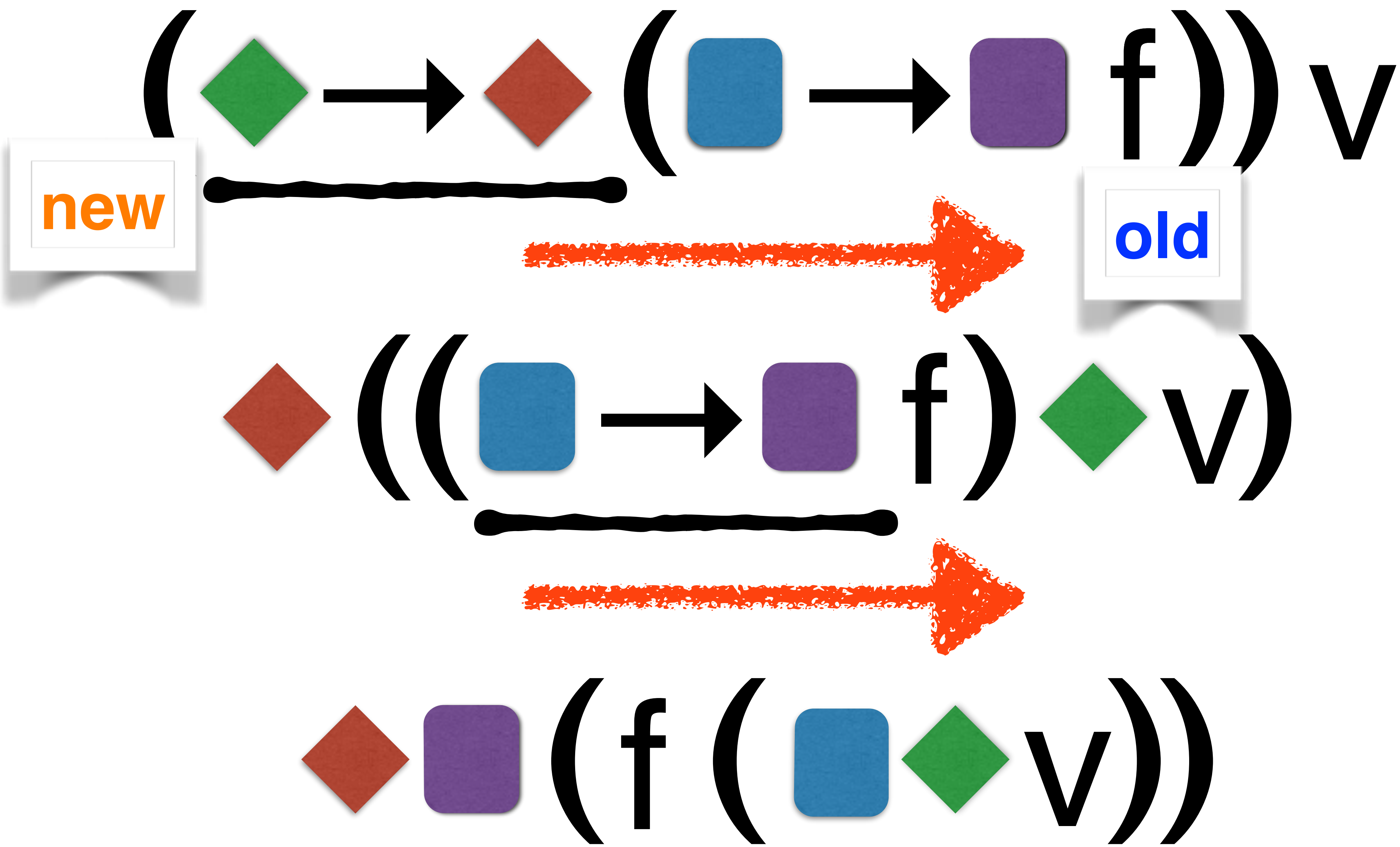
Merging function proxies



Merging function proxies

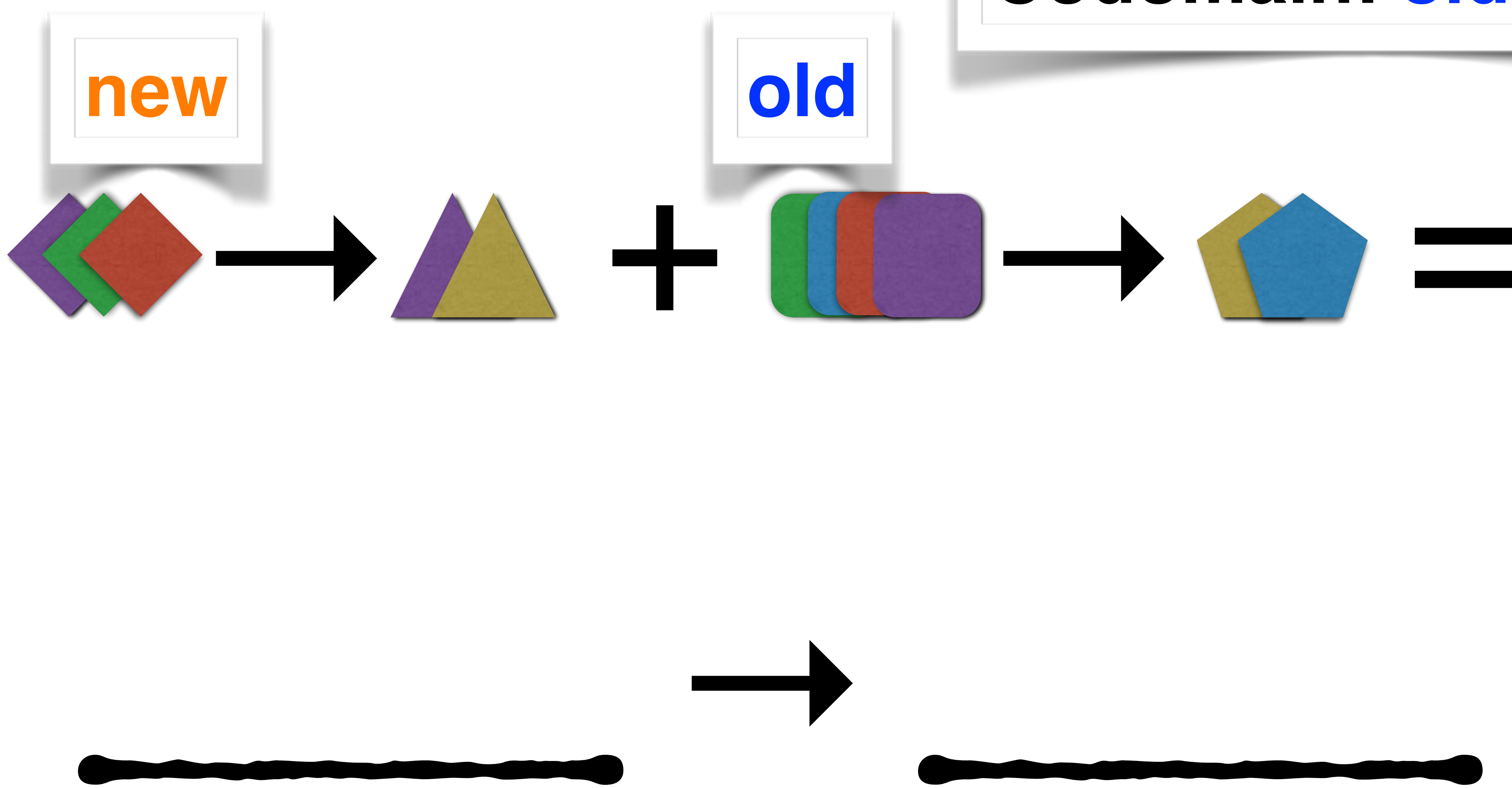


Merging function proxies



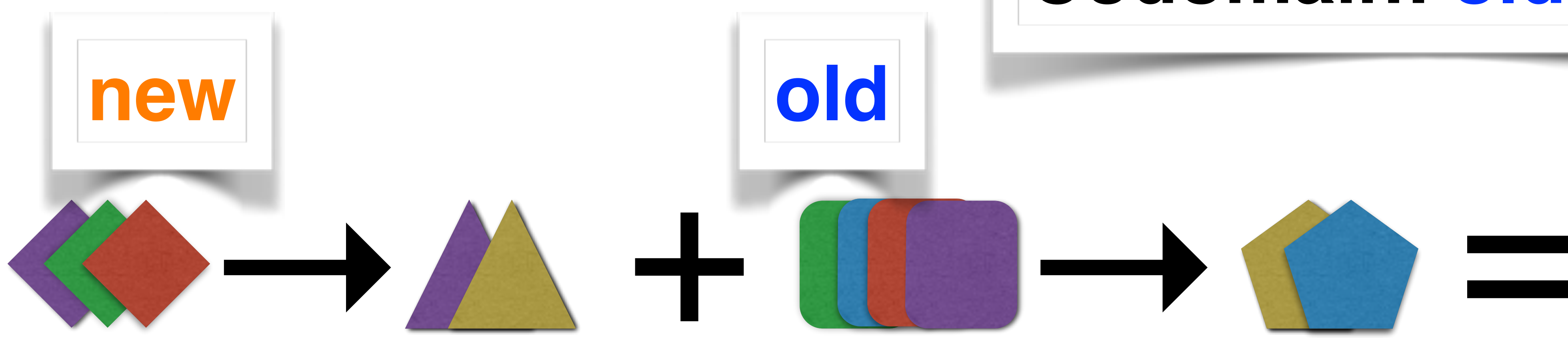
Merging function checks

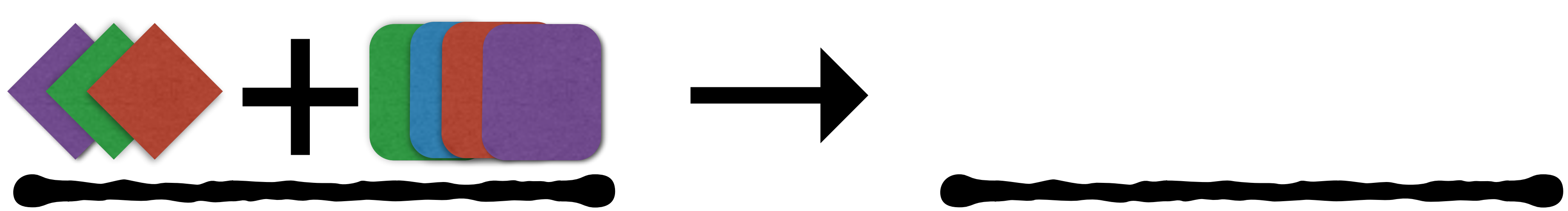
Domain: new to old
Codomain: old to new



Merging function checks

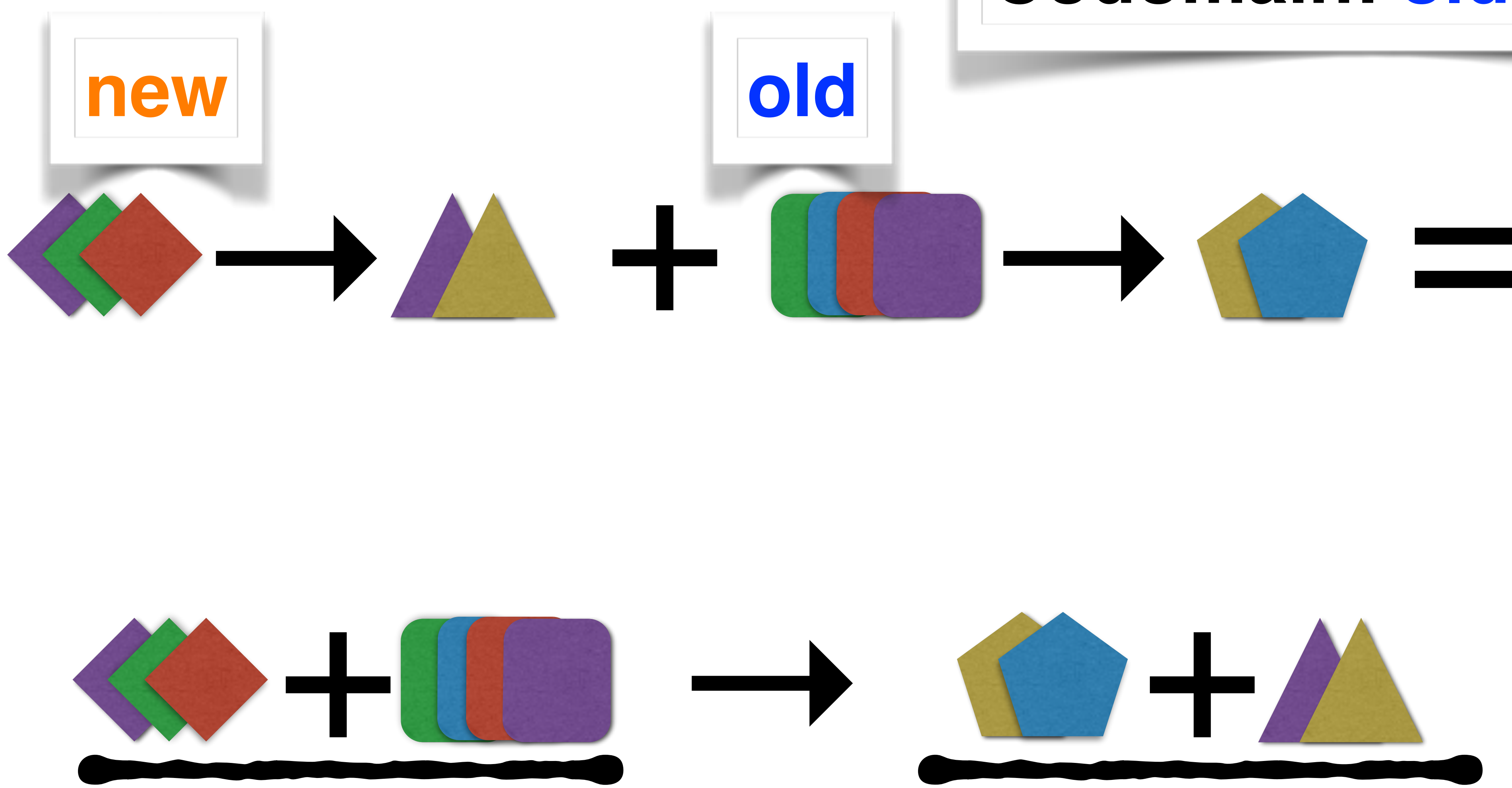
Domain: new to old
Codomain: old to new





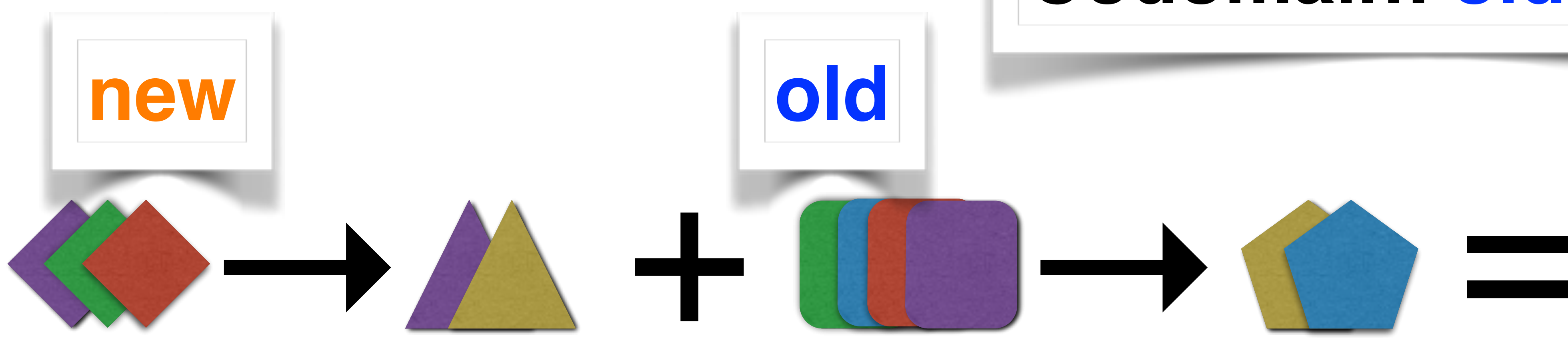
Merging function checks

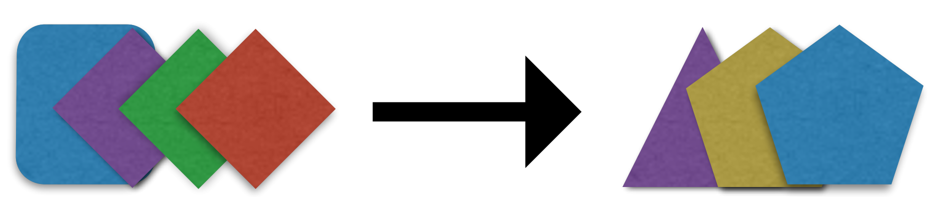
Domain: new to old
Codomain: old to new



Merging function checks

Domain: new to old
Codomain: old to new





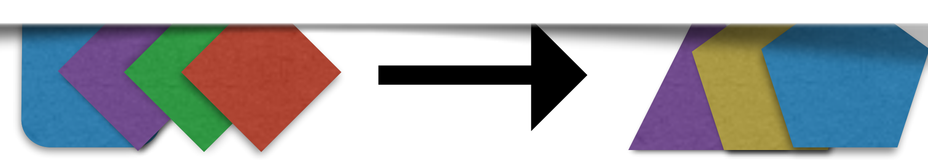
Merging function checks

: new to old
in: old to new

Go from right to left

Drop redundant checks

Contravariance



Proofs

Soundness

Classic semantics
and
space-efficient semantics
behave identically

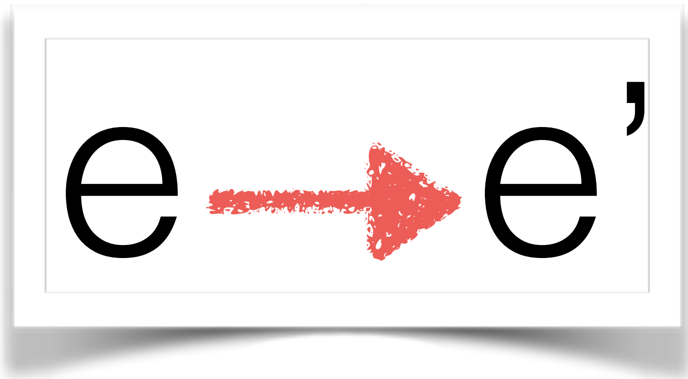
Congruence lemma

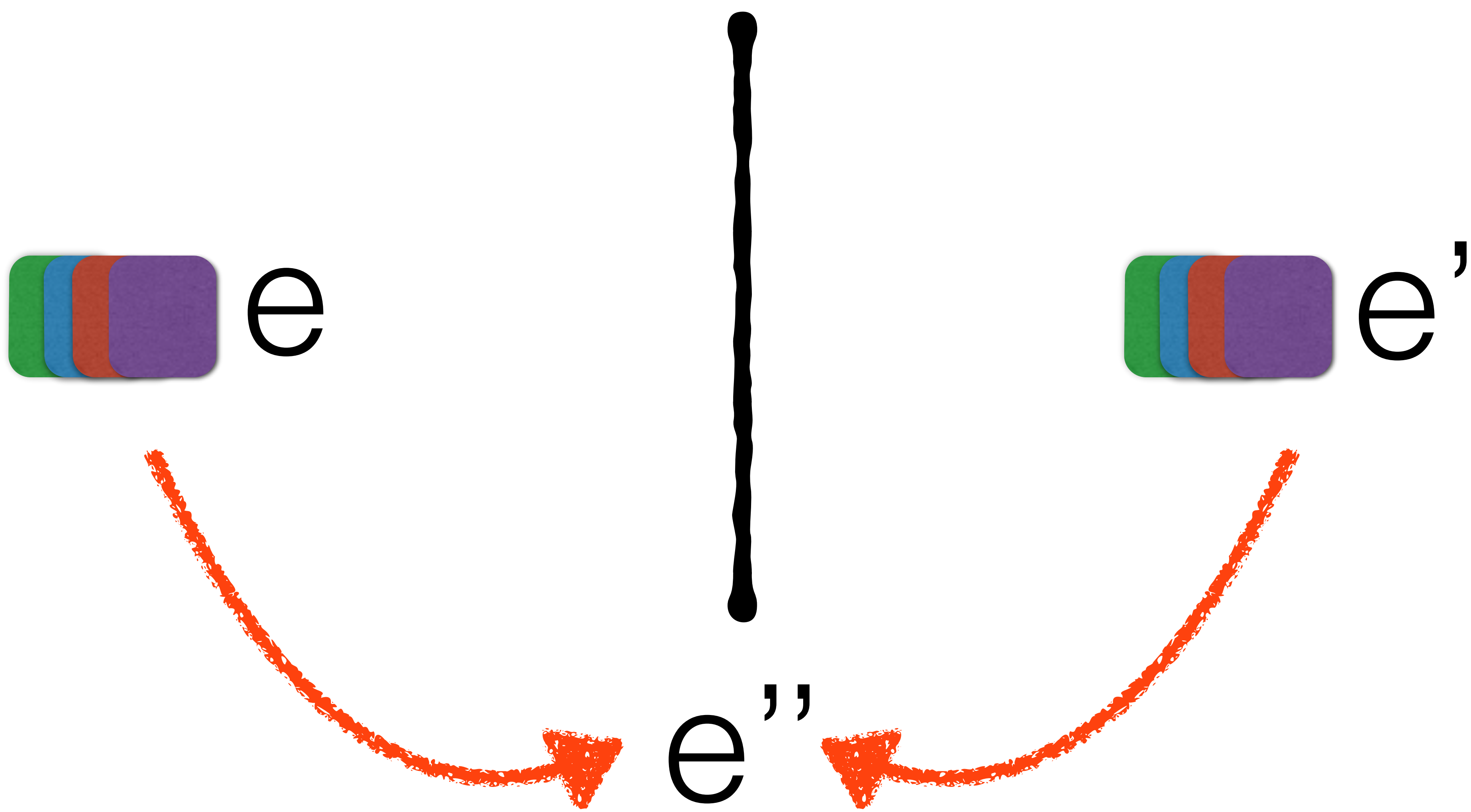
$$e_1 \longrightarrow_{\mathsf{E}} e_2$$

$$\Downarrow$$

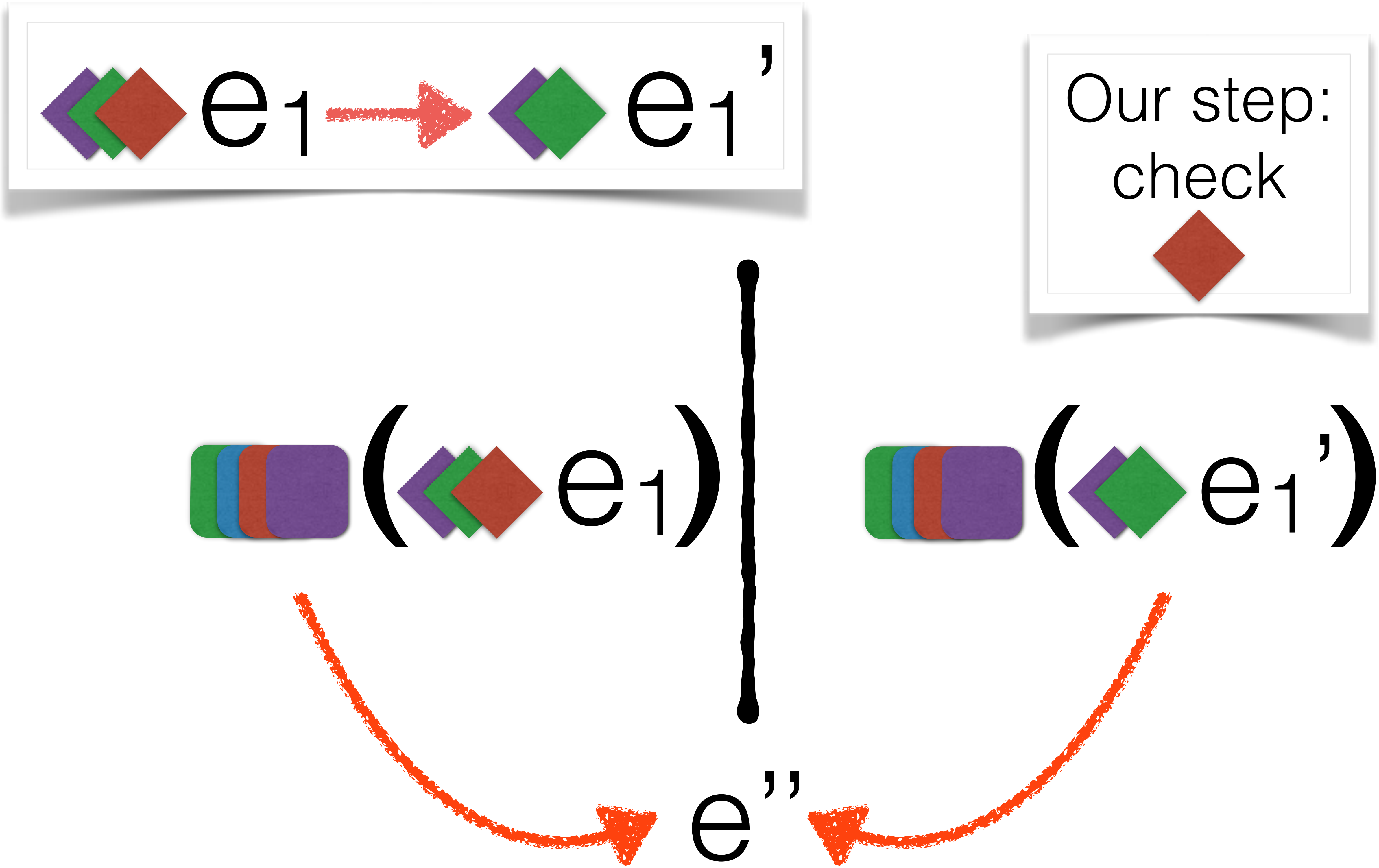
$$\begin{tikzcd}
\langle T_1 \overset{\bullet}{\Rightarrow} T_2 \rangle^l \; e_1 \arrow[dr, "*"', "\mathsf{E}"] & & \langle T_1 \overset{\bullet}{\Rightarrow} T_2 \rangle^l \; e_2 \arrow[dl, "*", "\mathsf{E}"'] \\
& \mathsf{result}_{\mathsf{E}} \; e &
\end{tikzcd}$$

Congruence lemma

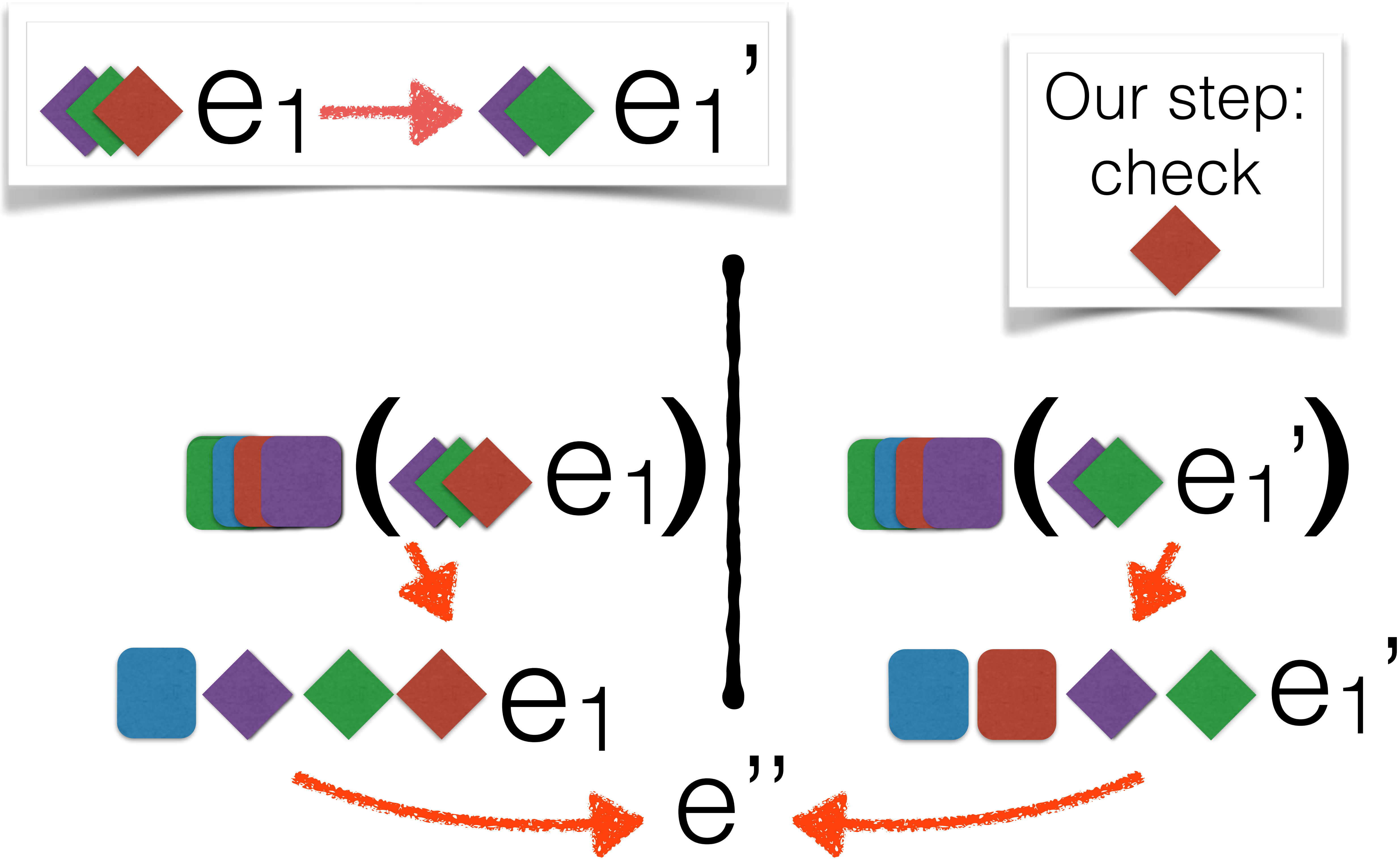




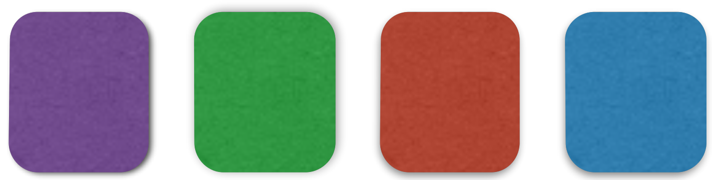
Congruence lemma

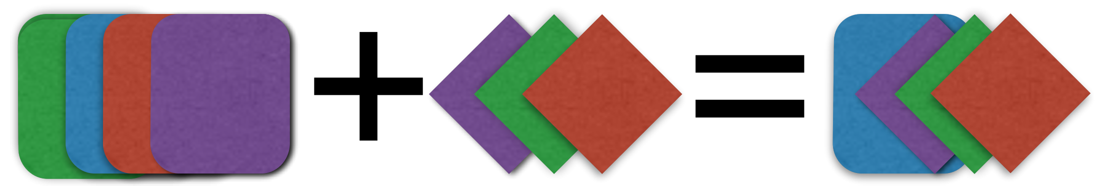


Congruence lemma



Outlook

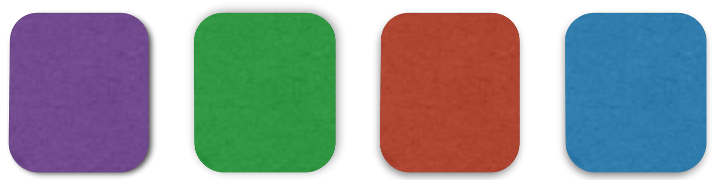
Types: 

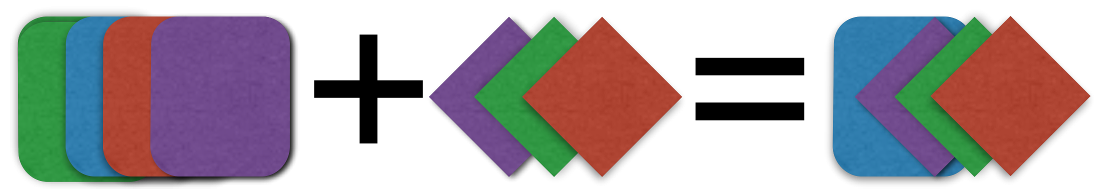


- Use **coercions**, not casts
- Merge **redundant** checks

Outlook

- Can we scale to **dependency**?

Types: 

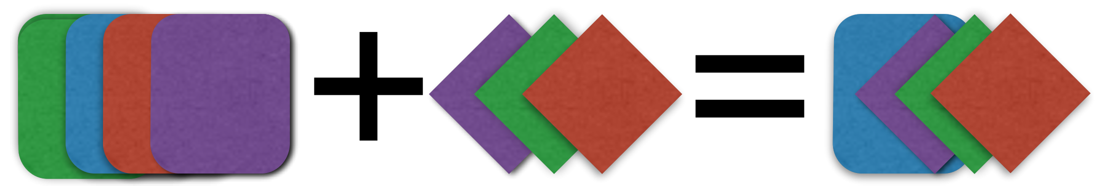


- Simple types—**finite number**
- Dependent types—**infinite number**

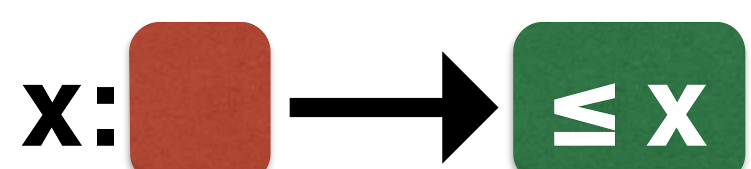
Outlook

- Can we scale to **dependency** and **effects**?

keep
lowest
check



- Idea: **partial orders**/**lattices**



Space-Efficient Manifest Contracts

Michael Greenberg
Princeton University
POPL 2015